6502
Anwendungen

Rodnay Zaks

SYBEX

6502
Anwendungen

6502 Serie - Band II

RODNAY ZAKS

6502 ANWENDUNGEN

ANMERKUNGEN

Der Verlag hat alle Sorgfalt walten lassen, um vollständige und akkurate Information zu publizieren. SYBEX-Verlag übernimmt keine Verantwortung für die Nutzung dieser Informationen, auch nicht für die Verletzung von Patent-, Lizenz- und anderen Rechten Dritter, die daraus resultieren. Es ist keine Lizenz von Herstellern an SYBEX erteilt worden und es sei insbesondere darauf hingewiesen, daß Hersteller Ihre Schaltpläne ändern ohne die breite Öffentlichkeit davon zu unterrichten. Technische Charakteristika und Preise können einem rapiden Wechsel ausgesetzt sein. Für die neuesten Technischen Daten ist es daher empfohlen, die Angaben der Hersteller zur Hand zu nehmen.

APPLE ist ein eingetragenes Warenzeichen von Apple Inc., USA.
CBM und PET sind eingetragene Warenzeichen von Commodore Inc., USA.
SYM ist ein eingetragenes Warenzeichen von SYNERTEK Systems Inc., USA.
AIM65 ist ein eingetragenes Warenzeichen von Rockwell International, USA.

Originalausgabe in Englisch.
Titel der englischen Ausgabe: 6502 Applications

Deutsche Übersetzung und Bearbeitung durch: Elmar Compans, Karlsruhe

Technische Zeichnungen: B. Janoff und R. Woodbury
Umschlag: Daniel Le Noury
Satz und Layout: tgr – typo-grafik-repro gmbh., remscheid
Gesamtherstellung: Druckerei Hub. Hoch, Düsseldorf

ISBN 3-88745-014-0
1. Auflage 1983

Printed in Germany

Vorwort

In diesem Buch werden praktische Anwendungen für den Mikroprozessor 6502 vorgestellt. Es werden elementare Kenntnisse der Mikroprozessor-Programmierung vorausgesetzt, wie sie Band I der 6502-Serie vermittelt (*Programmierung des 6502*, ISBN 3-88745-011-6). Das Verständnis der Programmierung des Mikroprozessors selbst ist lediglich ein notwendiges Requisit für die eigentliche Programmierung eines Mikrocomputers mit seinen angeschlossenen Geräten. Als nächstes muß man lernen, wie man Anwenderprogramme schreibt, die die Ein/Ausgabetore und andere Möglichkeiten eines vorhandenen Systems einbeziehen. Diesem Problem widmet sich das vorliegende Buch. Es stellt die Techniken und Programme vor, die bei den typischen Anwendungen benötigt werden, und setzt dabei die Ein/Ausgabebausteine ein, die in 6502 Mikrocomputern verfügbar sind.

Die in diesem Buch vorgestellten Programme benötigen nur ein Minimum an Hardware, um wirkungsvoll eingesetzt zu werden. Dem Leser sei daher dringend empfohlen, die hier vorgestellten Methoden und Techniken an der entsprechenden Hardware zu erproben. Die möglichen Anwenderplatinen werden detailliert beschrieben. Die Programme sind auf jeden 6502 Mikrocomputer anwendbar, etwa auf den KIM, den SYM, den AIM65, alle Geräte von COMMODORE, nach einer entsprechenden Erweiterung auch auf APPLE sowie auf andere. Viele Programme sind auf einem oder mehreren dieser Mikrocomputer direkt lauffähig, während für andere geringfügige Anpassungen vorgenommen werden müssen. Die Methoden und Techniken sind allen jedoch gemeinsam.

Die in diesem Buch vorgestellten Anwenderprogramme erlauben dem Leser beispielsweise den Bau einer kompletten Alarmanlage für sein Haus, die auch einen Feuermelder und vieles andere umfaßt, den Bau einer elektronischen Orgel, einer Geschwindigkeitsregelung für Elektromotoren, einer 24-Stunden-Uhr, einer simulierten Verkehrssteueranlage, eines Morse-Generators, einer Industriesteuerung zur Temperaturüberwachung, die einen programmierten Analog zu Digital-Wandler umfaßt, und anderes mehr.

Ziel des vorliegenden Buches ist es, dem Leser all die praktischen Fähigkeiten zu vermitteln, die er bei der Anwendung seines 6502 Mikrocomputers benötigt. Im Rahmen der 6502 Serie geht diesem Buch die *Programmierung des 6502* voraus, ihm folgt *6502 Advanced Programming,* das zur Zeit in der englischen Ausgabe vorliegt.

Die 6502 Serie

Band I PROGRAMMIERUNG DES 6502
Band II 6502 ANWENDUNGEN
Band III – 6502 ADVANCED PROGRAMMING
(nur in englischer Ausgabe erhältlich)

Danksagung
Viele Personen haben einen Beitrag zu diesem Buch, seiner Überprüfung, Entwicklung und Verbesserung, beigetragen. Besonderer Dank gilt Pierre Le Beux, Daniel David, Jaff Lin, Eric Martinot, Tricourt und Eric Novikoff (ASM65 Assembler).

Elmar Compans möchte ich besonders für seine Überarbeitung in der deutschen Fassung danken. Insbesondere die Übertragung der meisten Programme in deutsche und lauffähige Fassungen wird zum Verständnis der Materie viel beitragen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wenn man programmieren lernt, ist das Verständnis der Arbeitsweise des eigentlichen Mikroprozessors nur das erste Problem, das zu lösen ist. Diesem Problem wendet sich unser Buch *„Programmierung des 6502“* zu. Das nächste Problem ist das Erlernen einer effektiven Programmierung, wobei Ein/Ausgabe-Bausteine verwendet werden, die mit der Prozessorplatine verbunden sind. Das ist das Anliegen dieses Buches. Natürlich kann kein Buch vollständig alle möglichen Bausteine abdecken. Daher wurde unter den wichtigen Ein/Ausgabebausteinen, die man in Verbindung mit dem 6502 gewöhnlich antrifft, eine Auswahl getroffen, und es werden Anwendungsprogramme vorgestellt, die wahrscheinlich auf die Mehrheit der Anwendungsfälle passen.

Als erstes werden Sie lernen, wie man einen PIO, einen „Parallel-Ein/Ausgabe-Baustein“ effektiv programmiert. Sie werden lernen, wie man Abfragetechniken oder Interrupts anwendet. Sie werden Impulse erzeugen, Sie lernen das Messen von Verzögerungszeiten und die Steuerung gewisser Ein/Ausgabebausteine, wie etwa Schalter, Relais oder komplexerer Bausteine wie Digital-Analog Wandler, Motoren usw. Sie werden auch lernen, wie man komplexere Ein/Ausgabebausteine verwendet, z. B. *programmierbare Zeitgeber*. Weitere Interfacebausteine werden für einfache Schaltungen vorgestellt werden, so daß Sie Ihre eigene Anwenderplatine aufbauen und damit arbeiten können.

Um das Programmieren effektiv zu erlernen, müssen Sie sich intensiv in der Praxis üben. In der Tat ist die Praxis der einzige Weg, ein geübter Programmierer zu werden. Um praktische Beispiele auszuprobieren, werden Sie einen Mikrocomputer benötigen, etwa den KIM, den SYM, den AIM65, den PET, den CBM, den VC20, den APPLE, oder irgendeinen

anderen 6502 Mikrocomputer. Da alle Mikrocomputer normalerweise mindestens einen PIO (oft zwei) und mindestens zwei Zeitgeber (manchmal mehrere) besitzen, sollten die meisten der in diesem Buch vorgestellten Programme auf allen diesen Mikrocomputern mit gegebenenfalls durchzuführenden kleinen Änderungen ablauffähig sein.

Die zusätzliche Hardware, die Sie benötigen, um spezielle Programme laufen zu lassen, wird in den Kapiteln 4, 5 und 6 ausführlich diskutiert werden. Der Aufwand ist gering und die Teile sind leicht zu beschaffen. Insbesondere werden in den Kapiteln 4, 5 und 6 Vorschläge für derartige Anwenderplatinen beschrieben, die Sie mit gängigen Bauteilen preisgünstig nachbauen können. Durch die Verwendung Ihres Mikrocomputers zusammen mit der Anwenderplatine werden Sie in der Lage sein, alle Programme dieser Kapitel auszuprobieren. Es empfiehlt sich, den Selbstbau als einen ersten Schritt in die Praxis hin zu betrachten.

Unerläßlich ist dies jedoch nicht. Auch durch bloßes Lesen dieses Buches werden Sie die gesamte Standardtechnologie erlernen. Wenn Sie jedoch über dieses Stadium hinauswollen, wird die Praxis unverzichtbar.

Das Anschließen des Mikroprozessors an seine Umwelt

Das Anschließen des eigentlichen Mikroprozessors an seine Umwelt beinhaltet zunächst den Aufbau einer Standard-Mikrocomputer-Platine und anschließend den Anschluß an schon vorhandene Geräte. Es werden sowohl Hardware- als auch Softwareinterfaces benötigt, um die eigentlichen Geräte an die Platine anschließen zu können. Dieses Buch wird sowohl die Hardware-Komponenten als auch die Programme, die für die am häufigsten verwendeten Geräte gebraucht werden, im Detail vorstellen. Um industrielle Programme zu entwickeln, die gewöhnlich teure Anlagen wie z. B. Ampelanlagen steuern, werden wir auf der Anwenderplatine diese Anlagen simulieren, beispielsweise mit LEDs. Würde man das Programm mit einer wirklichen Ampelanlage kombinieren, so würde sich im allgemeinen nur die Interface-Hardware ändern. Das Programm jedoch bliebe im wesentlichen das selbe. Daher ist das Wissen, das Sie sich aneignen, anwendbar auf Situationen in Ihrer tatsächlichen Umwelt.

Das pädagogische Konzept

Wenn Sie dieses Buch lesen, werden Sie „durch die Praxis lernen". Jedes Programm wird im Detail vorgestellt: sein Zweck, sein Flußdiagramm, das Hardware-Interface, die Schaltungen, das Programm selbst und die vollständige Analyse der verwendeten Methoden. Jedes Kapitel ist im wesentlichen in sich abgeschlossen. Beispielsweise ist es nicht nötig, alle Besonderheiten der PIOs des Kapitel 2 zu verstehen, nur um Kapitel 3 zu lesen. Für ein umfassendes Verständnis ist abschnittweises Vorgehen beim Studium jedoch zu empfehlen. Im Kapitel 2 werden alle gängigen

parallelen E/A-Bausteine (E/A = Ein/Ausgabe), die in 6502-Systemen vorkommen (vom 6520 bis zum 6532) nacheinander vorgestellt. Da alle existierenden 6502 Mikrocomputer derzeit diese Standardbausteine verwenden, sollten alle Leser, die mit ihnen noch nicht vertraut sind, dieses Kapitel studieren.

Kapitel 3 behandelt den „Standard-6502-Mikrocomputer" und einige wohlbekannte existierende Varianten: KIM, SYM, AIM65, PET/CBM/VC20 und APPLE (es gibt natürlich noch weitere). Die meisten der in diesem Buch vorgestellten Beispiele laufen direkt auf dem SYM und mit geringfügigen Änderungen auch auf KIM und anderen Mikrocomputern.

Kapitel 4 führt in die Standardtechniken bei Anwendungen mit Anschluß einfacher Geräte ein: Relais, Schalter, Lautsprecher. Die erste Anwenderplatine wird für Anwendungen vom Morse-Generator bis hin zum Telefonwähler eingesetzt.

Kapitel 5 befaßt sich mit komplexeren Heim- und Industrieanwendungen. Die zweite Anwenderplatine wendet sich Problemen zu, die von der simulierten Verkehrsüberwachung und einer Analog-Digital Wandlung bis hin zur kompletten Alarmanlage oder zum elektronischen Klavier reichen.

Im Kapitel 6 werden dann preisgünstige reale Peripheriegeräte an den Mikrocomputer angeschlossen: Vom Lochstreifenleser bis hin zur Tastatur und zum Drucker.

Schließlich geben wir im Kapitel 7 eine Zusammenfassung und eine Synthese.

Weiterhin werden Sie im Anhang A einen kompletten BASIC-Assembler für den 6502 finden, der Ihnen die Entwicklung komplexerer Programme, die einen Assembler erfordern, erleichtern wird.

Auf der folgenden Seite finden Sie einen Standardvordruck, der dazu gedacht ist, Ihnen das Schreiben von Programmen für die CPU 6502 zu erleichtern.

SYBEX **Standard Programm-Vordruck**

Programmname ____________________

Programmierer ____________________ Datum __/__/__ Blat __/__

hexadezimal				symbolische Assembler-Anweisungen			
Adresse	1	2	3	Marke	Mnem	Operand	Kommentar

Bild 1.1: Standard Programm-Vordruck

Kapitel 2

Die Ein/Ausgabe-Bausteine

Einleitung

Wir werden in diesem Buch eine Vielzahl von Ein/Ausgabebausteinen an einen 6502-Mikrocomputer anschließen, um praktische Mikrocomputeranwendungen zu realisieren. Es ist daher wesentlich, die Ein/Ausgabe-Komponenten eines 6502-Systems zu verstehen. Für denjenigen Leser, der mit den Grundbegriffen oder den Standardtechniken (wie etwa Abfragemethoden) noch nicht vertraut ist, empfiehlt es sich, diese im vorangehenden Werk dieser Serie nachzulesen (*„Programmierung des 6502“*).

Wir werden in diesem Kapitel systematisch alle parallelen Ein/Ausgabebausteine besprechen, die praktisch jeden 6502-Mikrocomputer mit den nötigen Ein/Ausgabe-Möglichkeiten ausstatten. Es ist unerläßlich, wenigstens zu verstehen, wie ein „PIO“ (beispielsweise der 6522) arbeitet, bevor man zu den Kapiteln mit den Anwendungen übergeht. Die genauen Einzelheiten des Einsatzes eines Zeitgebers oder anderer exotischer Bausteine (wie etwa Schieberegister) sind für ein erstes Verständnis nicht wesentlich und können übersprungen werden. Ebenso sind die Feinheiten und Formate der verschiedenen internen Register des 6520, 6522, 6530 und 6532 nicht so wichtig, daß man sie stets parat haben muß. Sie werden an dieser Stelle mehr als Nachschlagemöglichkeit für spätere Kapitel eingeführt.

Wir schlagen daher vor, daß man wenigstens einen der Abschnitte über PIO's sorgfältig studiert, beispielsweise das über den 6520 oder den 6522. Sie sollten jedoch Ihr Gedächtnis nicht mit Details belasten, und sich nur auf die Funktionsweise konzentrieren. Fast jede Anwendung wird von einem PIO Gebrauch machen, d. h. also von einem der in diesem Kapitel vorgestellten Bausteine.

Außer diesen Chips werden die meisten Mikrocomputer noch einige andere spezielle Bausteine für Ein/Ausgabe-Interfaces verwenden, wie z. B. Kassetten-Interfaces oder CRT-Interfaces. Für Details dieser Interfaces sei der interessierte Leser an die Datenblätter der Hersteller oder das Buch *„Mikroprozessor Interface Techniken“* (ISBN 3-88745-012-4) verwiesen.

Grundlegende Definitionen

In diesem Abschnitt rufen wir uns diejenigen Begriffe ins Gedächtnis zurück, die wir in diesem Kapitel benutzen werden.

Die drei wesentlichen Ein/Ausgabe-Komponenten nahezu eines jeden Mikrocomputers sind der „PIO“, der „UART“ und der „Zeitgeber“. Lassen Sie uns diese untersuchen.

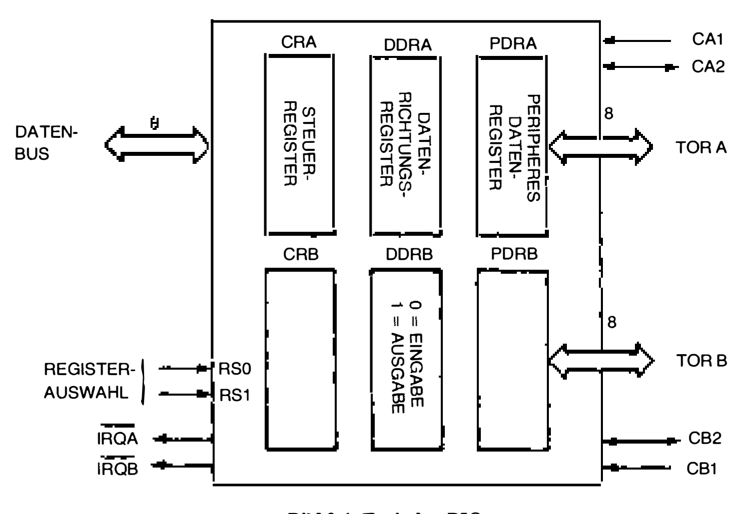


Bild 2.1: Typischer PIO

Der PIO

Der „PIO“ oder auch „Parallel-Ein/Ausgabe-Baustein“ (engl. „parallel input-output chip“) ist eine Komponente, die wenigstens zwei parallele 8-Bit Tore vorsieht. Für gewöhnlich ist bei einem PIO die Richtung einer jeden Leitung eines jedes Tores programmierbar. Die Richtung jeder Leitung ist normalerweise durch den Inhalt eines „Daten-Richtungs-Registers“ (DDR) (engl. „data-direction register“) festgelegt, das zu jedem Tor gehört. Zum Beispiel ist stets, wenn ein bestimmtes Bit des Daten-

richtungsregisters „0“ ist, die entsprechende Leitung des Tores auf Eingabe geschaltet. Vor Verwendung des PIO muß der Programmierer also zuerst den Inhalt des Datenrichtungsregisters jedes Tores laden, um zu definieren, in welcher Richtung die einzelnen Leitungen betrieben werden sollen. Besondere zusätzliche Bedingungen können vom Hersteller vorgegeben sein, beispielsweise die Einschränkung, die Leitungen nur in Vierergruppen in einer Richtung programmieren zu können, oder andere spezielle Zuweisungen, die nur einzelne Bits, etwa Bit 6 und Bit 7, betreffen. Einige solche Beschränkungen werden uns bei der Besprechung der verschiedenen Bausteine in diesem Kapitel begegnen. Das interne Blockschaltbild eines „Standard-PIO“ zeigt Bild 2.1. Die beiden Puffer für die Tore A und B erscheinen rechts im Bild. Das zum jeweiligen Tor gehörende Datenrichtungsregister steht links vom Puffer. Zusätzlich sind in diesem vereinfachten Diagramm noch zwei Steuerregister aufgeführt. Das Steuerregister dient dazu, die Funktion von Steuersignalen festzulegen, die vom PIO erzeugt werden. Insbesondere muß durch das Steuerregister das Verfahren des Quittungsbetriebs (engl. „handshake“) festgelegt und kontrolliert werden, ferner, ob die Steuersignale Flags verändern oder Interrupts auslösen sollen, und beispielsweise auch, ob 0/1-Übergänge oder 1/0-Übergänge verwendet werden sollen. Im allgemeinen wird der Programmierer den Inhalt der Steuerregister vor jeder Benutzung der Steuerleitungen zuerst festlegen müssen. Weiter wird der Programmierer den Inhalt des Steuerregisters abfragen, um festzustellen, ob ein interner Interrupt oder eine andere spezielle Bedingung vorliegt (Status-Information).

Zusätzlich zu den zwei Datentoren sollte ein PIO auch noch über Steuerleitungen verfügen, die automatischen Quittungsbetrieb mit Peripheriegeräten erlauben. Diese Steuerleitungen sind auf der rechten Seite des Standard-PIO im Bild 2.1 gezeigt. Für Tor A heißen sie jeweils CA1 und CA2, für Tor B CB1 und CB2.

Als ein Beispiel für Quittungsbetrieb könnte das externe Peripheriegerät ein „Daten bereit“-Signal an CA1 legen. Der PIO würde mit einem „Daten-Anforderung“-Signal an CA2 antworten. Zusätzlich sollte, wenn das „Daten bereit“-Signal an CA1 empfangen wird, im Steuerregister ein Flag gesetzt werden und es könnte ein externer Interrupt erzeugt werden, um den 6502 über diesen Vorgang zu informieren. Dies ist ein typisches einfaches Beispiel für die Abfolge der Steuersignale, die für Quittungsbetrieb nötig sind. Vieles davon läuft automatisch im Innern des Standard-PIO ab, und die Optionen werden durch den Inhalt des Steuerregisters festgelegt. Die Einzelheiten werden wir dann für jeden PIO gesondert beschreiben.

Der Zeitgeber

Bei den meisten Anwendungen ist eine grundlegende Forderung, bestimmte *Verzögerungszeiten* erzeugen zu können. Verzögerungszeiten können softwaremäßig gemessen werden, oder aber hardwaremäßig mit Zeitgebern. Solange in dem System keine Interrupts verwendet werden,

können Verzögerungszeiten normalerweise bequem mit Programmschleifen erzeugt werden (für eine genauere Beschreibung dieses Verfahrens siehe *„Programmierung des 6502“*). In komplexeren Situationen, oder wenn Hardwareinterrupts auftreten können, ist jedoch die Verwendung von einem oder mehreren Zeitgeberbausteinen zur Erzeugung oder Messung von Zeitverzögerungen wünschenswert.

Benutzung des Zeitgebers für Ausgabeoperationen

In seiner einfachsten Form ist ein Zeitgeberbaustein ein mit einem Register (8 oder 16 Bit) ausgestatteter Zähler. Beim Betrieb im Ausgabemodus wird das Zählerregister vom Programm mit einem vorgegebenen Wert geladen. Dann wird ein Startimpuls gegeben und der Zeitgeber beginnt zu zählen. Die meisten Zeitgeber verwenden den systeminternen Taktgeber, aber nicht alle. Meist haben diese Taktgeber eine Frequenz von 1 MHz, das ergibt 1 µsec-Impulse. Die im Zählerregister abgelegte Zahl wird bei jedem Taktimpuls um 1 erniedrigt (dekrementiert). Hatte der Registerinhalt den Wert N, so erreicht der Dekrementiervorgang nach N Taktimpulsen den Registerinhalt 0. Nimmt man 1 µsec-Impulse an, so tritt dies nach genau N µsec ein. Jedesmal, wenn der Zähler 0 erreicht, wird ein Steuersignal erzeugt, das im Zeitgeber ein internes Statusflag setzt und/oder einen externen Interrupt erzeugt. Je nach erforderlicher Genauigkeit wird das Programm entweder die gerade laufenden Zeitgeber abfragen, oder aber einen Hardwareinterrupt annehmen. Typische Programme werden in diesem Kapitel vorgestellt.

Wäre der Zeitgeber mit einem einzigen 8-Bit Register ausgerüstet, könnte er nur von 1 bis 256 zählen. Die größtmögliche Verzögerungszeit wäre bei Verwendung eines Standard-Taktgebers somit nur 256 µsec. Für die meisten Anwendungen ist diese Zeit zu kurz. Selbstverständlich wäre es möglich, die am Ende der 256 µsec erzeugten Interrupts in einem Speicher aufzusummieren und dann zu testen, ob der Inhalt dieses Speichers einen gewissen Endwert schon erreicht hat. Jedoch hätte dies ungenaue Zeitmessung und eine schwerfällige Programmierung zur Folge. Ein nur mit einem 8-Bit Register augestatteter Zeitgeber wäre daher nicht ausreichend. Man verwendet zwei Techniken, diese Einschränkung zu überwinden. Die vom Konzept her einfachste ist es, für den Zähler ein 16-Bit Register zu verwenden. Damit kann der Zähler von 1 bis 64K zählen, d. h. Verzögerungszeiten von 1 µsec bis zu 65536 µsec (maximal also ca. 65 msec) darstellen. Dies ist dann in der Tat für die meisten Anwendungen ausreichend. Jedoch erfordert diese Technik, daß der Zeitgeber in mindestens zwei Schritten geladen wird, da der Datenbus nur 8 Bit breit ist. Das Programm muß also zuerst die eine Hälfte des Zählerregisters laden, dann die andere Hälfte. Das ist umständlich.

Die andere Technik, mit der man Verzögerungszeiten innerhalb größerer Grenzen erzeugen kann, ist die Verwendung interner Dividierer im Zeitgeber. Ein solcher Baustein ist mit beispielsweise vier Registern ausgerü-

stet. Wird das erste Register verwendet, so wird die erzeugte Verzögerungszeit in Taktzyklen ausgedrückt sein (typischerweise 1 µsec). Wird das zweite Register verwendet, wird die Einheit der Verzögerungszeit das 8-fache der Taktzykluszeit sein; mit dem dritten wird sie das 64-fache und mit dem vierten das 1024-fache der Taktzykluszeit sein (bei Annahme eines 1 MHz-Taktgebers also ca. 1 msec). Diese Technik ist für den Programmierer ein wenig bequemer und bietet die Möglichkeit, den Zeitgeber in nur einem Schritt zu laden, trotz der Anwendung über große Zeitbereiche. Der interne Aufwand innerhalb des Bausteins steigt jedoch beträchtlich.

Benutzung des Zeitgebers für Eingabeoperationen

Ein auf Eingabe geschalteter Zeitgeber kann zur Messung externer Pulsdauern oder der Zeit zwischen zwei aufeinanderfolgenden Impulsen verwendet werden. In diesem Fall ist der Anfangswert des Zählerregisters Null und der Zähler wird in jedem Taktintervall sein Register inkrementieren. Sobald der Meßvorgang beendet ist, wird ein internes Flag gesetzt oder es wird ein externer Interrupt erzeugt. Das Programm ist dann dafür verantwortlich, den Inhalt des Zählerregisters, der ja die Dauer des externen Ereignisses anzeigt, zu lesen.

Impulsketten

Ein Zeitgeber kann nicht nur zur Erzeugung oder Messung von Einzelimpulsen, sondern auch zur Erzeugung oder Zählung von Impulsketten verwendet werden. Wenn eine Verzögerung für nur einen Impuls erzeugt oder gemessen wird, nennt man das „Einzelbetrieb". Wird eine ganze Impulskette erzeugt, spricht man meist vom „Freilaufmodus". Zusätzlich kann eine Zahl von Optionen verfügbar sein, mit denen festgelegt wird, ob als Start- oder Stopsignal für den Zeitgeber ein 0/1- oder ein 1/0-Übergang benutzt werden soll, oder ob der Zeitgeber auf Schwellen anstatt auf Impulse reagieren soll. Ferner kann der logische Wert und das zeitliche Verhalten von Interruptflags vereinbart werden. Weiterhin sind die Bedingungen, unter denen interne Statusbits gesetzt und zurückgesetzt werden, gewöhnlich programmierbar. Wegen der großen Zahl möglicher Varianten hat jeder Zeitgeber ausgeprägte spezifische Charakteristiken und muß vor seinem Einsatz im Detail studiert werden.

Der UART

„UART" ist die Abkürzung für „Universal Asynchronous Receiver Transmitter", d. h. „universeller asynchroner Empfänger und Sender". Die wesentliche Funktion eines UART ist die Umwandlung serieller in parallele und paralleler in serielle Daten. Zusätzlich besitzt ein Standard-UART eine Reihe weiterer Möglichkeiten, die normalerweise für einen seriellen Datenaustausch mit externen Geräten nötig sind (z. B. Paritäts-

kontrolle oder Paritätserzeugung, Start/Stop-Bits). Die eigentliche Umwandlung besorgen interne Schieberegister. Solche Schieberegister finden auch in anderen Ein/Ausgabebausteinen Verwendung.

Existierende 65xx Ein/Ausgabe-Bausteine

Jeder 6502-Mikrocomputer benötigt eigentlich mindestens zwei PIOs und einen Zeitgeber. Typischerweise werden diese Funktionen durch eine Kombination der Bausteine 6520/6530 oder 6522/6532 dargestellt. Die Bausteine 6520 und 6530, die im folgenden beschrieben werden, sind die original von MOS Technology eingeführten Ein/Ausgabebausteine. Die CPU 6502 wird inzwischen jedoch auch von einigen anderen Herstellern gefertigt, wie etwa Synertek und Rockwell, und so wurden zusätzliche Bausteine der 65xx Familie eingeführt, wie etwa 6522/6532. Weitere Bausteine der 65xx Familie werden wahrscheinlich im Laufe der Zeit folgen.

Derzeit sind die wichtigsten Bausteine jedoch der 6520, 6530 sowie 6522, 6532. Diese vier wesentlichen E/A-Bausteine werden wir im folgenden vorstellen.

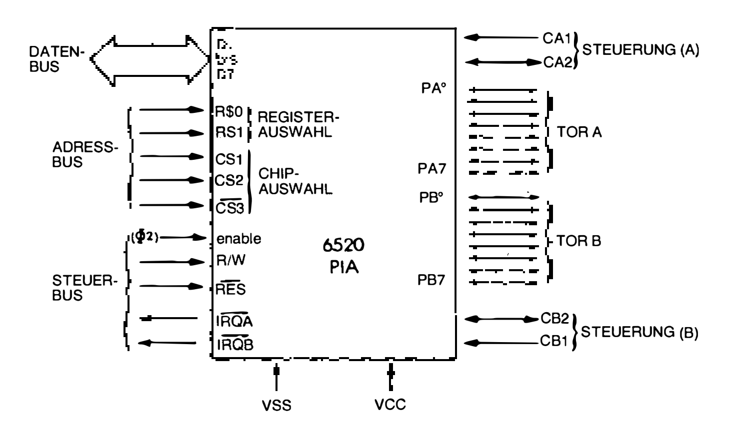


Bild 2.2: Der 6520 PIA

Der 6520 (PIA)

Der Baustein 6520 ist nach unserer Definition ein fast reiner „PIO". Er wurde als pinkompatibler Ersatz für den Motorola-IC M6820 konstruiert

und wird von seinem Hersteller ein „peripherer Interface-Adapter“, oder kurz „PIA“ genannt. Die nach außen geführten Signalleitungen des 6520 zeigt Bild 2.2, seine interne Architektur Bild 2.3.

In Bild 2.3 sehen wir, daß dieser Baustein zwei parallele Ein/Ausgabe-Tore (engl. „ports“) besitzt: Tor A und Tor B. Jedes Tor ist mit einem Puffer ausgerüstet. Jedoch sind die beiden Tore nicht völlig identisch und die Puffer arbeiten eigentlich nur als Ausgabe-Puffer, nicht jedoch als Eingabe-Puffer. Für jedes Tor ist ein Datenrichtungsregister („DDR“ = engl. data direction register) verfügbar. Es legt die Richtung jeder Leitung des betreffenden Tores fest. Eine „0“ in diesem Register legt für die zugeordnete Leitung die Eingabefunktion, eine „1“ die Ausgabefunktion fest. Die Wahl solcher Vereinbarungen geht auf Sicherheitsbetrachtungen zurück: jedesmal, wenn ein „reset“ angelegt wird, werden alle internen Register auf Null gesetzt und auch alle Datenrichtungsregister werden somit Null. Dies hat zur Folge, daß alle Leitungen als *Eingabe*kanäle interpretiert werden. Das führt zu einem sicheren Systemstart. Es kann kein externer Impuls erzeugt werden, solange nicht das Hauptprogramm gestartet wurde.

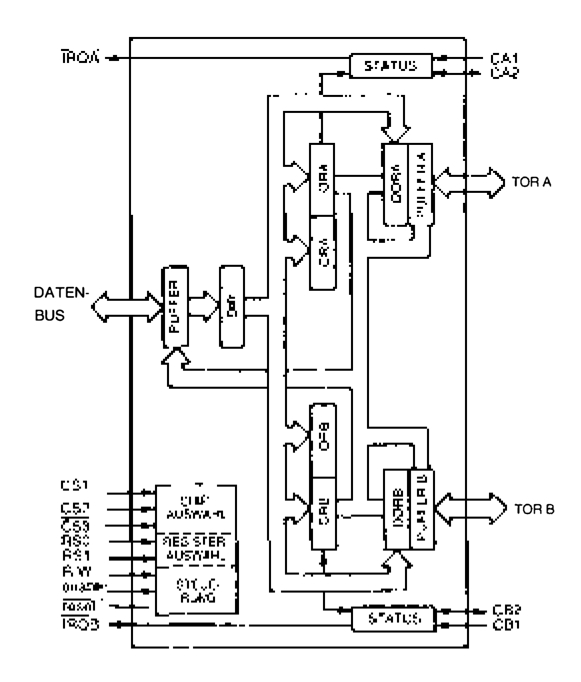


Bild 2.3: 6520, interne Architektur

Zusätzlich ist jedes Tor mit zwei weiteren Registern ausgerüstet, dem *Steuerregister* und dem *Ausgaberegister*. Die vom 6502 kommenden Daten werden zum *Ausgaberegister* (engl. output register, kurz ORA bzw. ORB) des jeweiligen Tores (A oder B) geleitet und dort gespeichert. Die Funktion des *Steuerregisters* (engl. control register, kurz CRA bzw. CRB) wird weiter unten erklärt. Es bestimmt die Rolle verschiedener Steuerfunktionen und enthält die Statusinformationen jedes Tores.

Schließlich ist jedes Tor noch mit zwei externen Steuerleitungen versehen, die im Falle des Tores A mit CA1 und CA2 benannt sind. CA1 ist eine unidirektionale Leitung in den 6520 hinein, CA2 ist bidirektional und kann somit sowohl als Eingabe- als auch als Ausgabekanal dienen.

Logisch sind die beiden Tore äquivalent und symmetrisch, wie schon Bild 2.3 zeigt. Es gibt jedoch praktische Unterschiede. Insbesondere ist das Treibervermögen von Tor B dem von Tor A überlegen und die Rolle der Steuersignale ist nicht völlig symmetrisch.

Mit einem Blick auf die linke Seite von Bild 2.3 oder auf Bild 2.2 sehen wir, daß der Puffer des 6520 den internen Datenbus an den System-Datenbus ankoppelt. Über zwei Leitungen können von der Einheit Interruptanforderungen erzeugt werden, falls dies in den Steuerregistern für Tor A oder Tor B so festgelegt ist. Sie heißen IRQA und IRQB. Schließlich müssen für den jeweiligen Baustein drei Chip-Auswahlleitungen angesteuert werden, sie heißen CS1, CS2 und CS3. Dieser Kunstgriff wurde von Motorola angewandt, um den bequemen geichzeitigen Direktanschluß von bis zu 8 ($=2^3$) getrennten Einheiten an den Datenbus zu erlauben, ohne daß ein externer Adreß-Dekoder nötig ist. In der Praxis kann die hohe Zahl von Chip-Auswahleingängen jedoch Nachteile mit sich bringen, (z. B. das Fehlen einer Register-Auswahl), die wir weiter unten erläutern werden. Die zwei vorhandenen Register-Auswahl-Eingänge sind mit dem Adreßbus verbunden. Sie heißen RS0 und RS1. Das bedeutet, daß der Baustein 6520 für den Programmierer wie *vier* Speicherplätze erscheint. Das mag überraschend erscheinen, da wir doch eben anhand von Bild 2.3 festgestellt haben, daß vier Register pro Tor existieren, insgesammt also *acht Register*. Wie aber kann man mit nur vier zur Verfügung stehenden Adressen acht Register adressieren? Dieses Problem wurde durch die begrenzte Zahl von Anschlüssen des Bausteins aufgeworfen. Deswegen übernimmt ein Bit des Steuerregisters, es ist das Bit 2, die Aufgabe, zwischen den beiden Registerblöcken auszuwählen. Wenn Bit 2 des Steuerregisters „0“ ist, so wird das Datenrichtungsregister des jeweiligen Tores angewählt. Ist es „1“, so wird das entsprechende Ein/ Ausgabe-Register angewählt.

Schließlich verfügt der Baustein noch über drei weitere Steuerleitungen: „R/W“ (engl. read or write, d. h. lesen oder schreiben), „enable“ (zulassen; normalerweise die zweite Phase des Taktes), und schließlich „reset“, die Rücksetzfunktion.

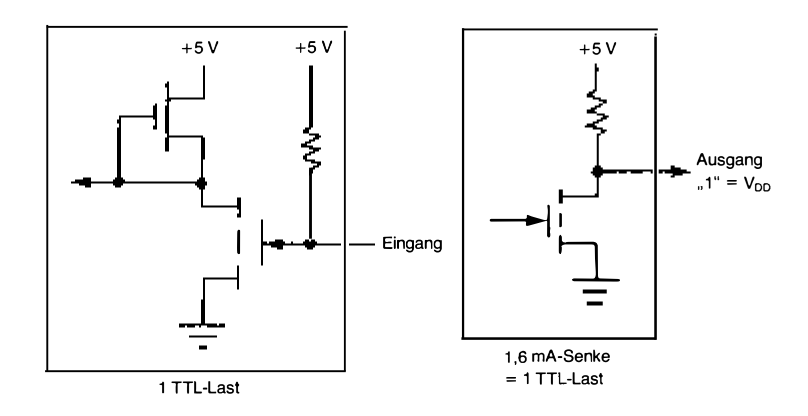


Bild 2.4: Puffer A

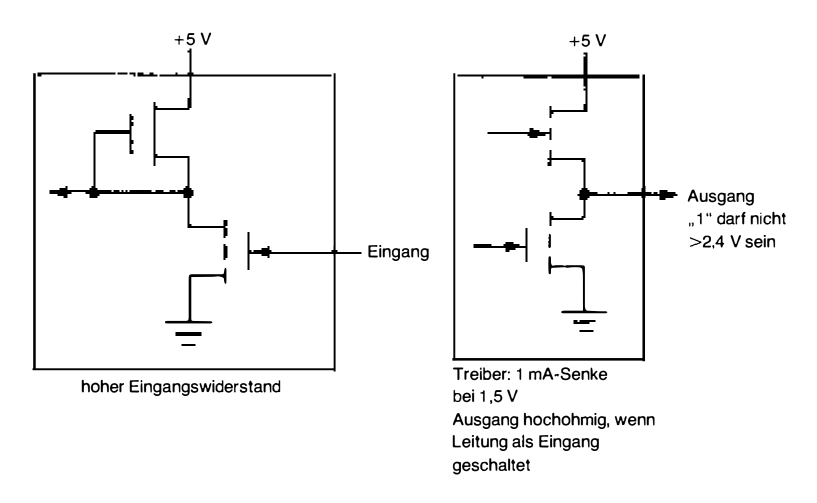


Bild 2.5: Puffer B

Unterschiede zwischen Tor A und Tor B

Obwohl die Tore A und B logisch gleichwertig sind, sind sie doch elektrisch ungleich. Die Puffer von Tor A verwenden *passive Ruheströme*, d. h. sie sind passiv mit Widerständen auf +5V gelegt. Sie können 1,6mA ziehen, damit sind sie in der Lage, einen Standard-TTL-Baustein zu trei-

ben. Im Tor B laufen sie im *push-pull*-Betrieb, d. h. zwei Transistoren arbeiten im Gegentakt (Bilder 2.4 und 2.5). Da es sich um *aktive* Schaltungen handelt, darf die Spannung für „1“ nicht höher als 2,4V zu sein (gegenüber V_{DD} beiTor A). Allerdings ist ihre Treiberfähigkeit überlegen (1mA bei 1,5V), so daß sie direkt an LED's (lichtemittierende Dioden) oder transistorisierte Schalter mit Darlington-Transistoren angeschlossen werden können. Außerdem geht der Ausgangspuffer von Tor B in einen *hochohmigen* Zustand, wenn Tor B als Eingabetor verwendet wird. Ein solcher Eingang hat einen hohen Widerstand (größer als ein Megaohm). Die Details des Puffers von Tor A zeigt Bild 2.4, die des Puffers von Tor B Bild 2.5.

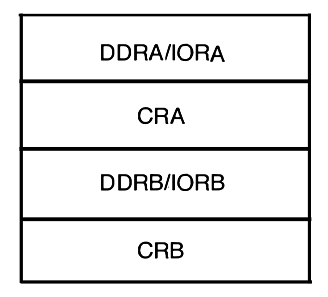


Bild 2.6: Register-Verzeichnis des 6520

Die internen Register

Wir wollen nun die speziellen Möglichkeiten und die Eigenarten des 6520 ein wenig mehr im Detail betrachten. Zuerst ist der 6520, wie schon erwähnt, mit 6 internen Registern ausgestattet: den zwei Puffern (die dieselben Adressen wie die Ausgaberegister haben), den zwei Datenrichtungsregistern und den zwei Steuerregistern. Da jedoch die Zahl der Anschlüsse des Gehäuses begrenzt ist, sind für diesen Baustein nur zwei Register-Auswahl-Leitungen verfügbar. Sie heißen RS0 und RS1. Das sich hieraus ergebende Registerverzeichnis zeigt Bild 2.6. Es zeigt, daß die Register DDRA/IORA sowie DDRB/IORB jeweils dieselbe logische Speicheradresse belegen. Das Steuerregister wird unabhängig adressiert. Die Unterscheidung zwischen DDRA und IORA (bzw. DDRB und IORB) wird von dem 6520 intern mit Hilfe von BIT 2 des Steuerregisters vorgenommen. Bild 2.7 zeigt die Registerauswahl. Wenn Bit 2 des Steuerregisters „0“ ist, wird DDR angesprochen, wenn es „1“ ist, wird IOR, also das Pufferregister angesprochen. Das Steuerregister ist das einzige Register, das mit RS0 und RS1 direkt adressiert werden kann, da es lo-

gisch notwendig ist, den Inhalt dieses Steuerregisters vor jeglichem Zugriff auf eines der anderen Register festzulegen.

RS1	RS0	CRA-2	CRB-2	Registerwahl
0	0	1	·	PufferA
0	0	0		DDRA
0	1	·	·	CRA
1	0	-	1	DDRB
1	1	-	·	CRB

Bild 2.7: 6520 Register-Auswahl

Das Schema macht deutlich, daß das Initialisieren dieses Bausteines etwas umfangreicher ist, als man das gerne hätte, und daß, falls das Programm abwechselnd Zugriff zum DDRA und zum IORA benötigt, zusätzliche Befehle eingefügt werden müssen, die jedesmal das BIT 2 des CRA ändern. Das ist in der Tat unbequem.

Das Steuerregister

Den Inhalt des Steuerregisters zeigt Bild 2.8. Daß das Bit 2 dieses Registers eine besondere Stellung einnimmt, wurde schon hervorgehoben: es trifft die Auswahl zwischen den Registern DDR und IOR des jeweiligen Tores. Die anderen Bits innerhalb des Registers legen Steuerfunktionen für die beiden Steuerleitungen jedes Tores fest und zwei Bits sind für Status- und Interruptinformationen reserviert. Die Funktionen der Steuerleitung CA2 werden von den Bits 3, 4, 5 gesteuert, wie Bild 2.9 zeigt.

7	6	5	4	3	2	1	0
IRQ1	IRQ2	CA/B2-Steuerung			DDRA/B Auswahl	CA/B1 Steuerung	

Bild 2.8: die Steuerregister der 6520

CRABIT 5	4	3	Betriebsart	Wirkungen
1	0	0	Quittungs-betrieb beim Lesen	• CA1 Interrupt-Eingangs-impuls setzt CA2 auf „1" • Lesebefehl für Tor A, setzt CA2 zurück auf „0"
1	0	1	Puls-Ausgabe	• Lesebefehl für Tor A, setzt CA2 für einen Takt auf „0" (Bestätigung)
1	1	0	manuelle Ausgabe	setzt CA2 auf „0"
1	1	1	manuelle Ausgabe	setzt CA2 auf „1"

Bild 2.9: 6520, Steuerung von CA2

CRBBIT 5	4	3	Betriebsart	Wirkungen
1	0	0	Quittungs-betrieb beim Schreiben	• CB1 Interrupt-Eingangs-impuls setzt CB2 auf „1" • Schreibbefehl für Tor B, setzt CB2 zurück auf „0"
1	0	1	Puls-Ausgabe	• Schreibbefehl für Tor B, setzt CB2 für einen Takt auf „0" (Bestätigung)
1	1	0	manuelle Ausgabe	setzt CB2 auf „0"
1	1	1	manuelle Ausgabe	setzt CB2 auf „1"

Bild 2.10: 6520, Steuerung von CB2

Die Funktionen der zwei Steuerleitungen von Tor B werden durch die Bits 3, 4 und 5 seines Steuerregisters festgelegt. Sie sind in Bild 2.10 angegeben. Die Bits 0 und 1 erledigen die Interruptsteuerung für die Eingänge CA1 und CB1. Das zeigt Bild 2.11.

CR BIT 1	0	Flanke des Eingangssignales	IRQ Ausgang
0	0	negativ	nicht zugelassen („1“)
0	1	negativ	zugelassen (wird auf „0“ gesetzt, wenn CRA Bit 7 durch CA1/CB1-Übergang gesetzt wird)
1	0	positiv	nicht zugelassen („1“)
1	1	positiv	zugelassen (wie oben)

Bild 2.11: Interrupt-Steuerung (Eingänge CA1 und CB1)

Die Benutzung des 6520

Nach Anlegen eines „reset“, der Rücksetzfunktion, sind die Inhalte aller Register Null. Der 6520 muß daher als erstes initialisiert werden, um die Ein- und Ausgabekanäle der beiden Tore festzulegen. Auch die Steuerfunktionen des Steuerregisters müssen festgelegt werden. Der 6520 sollte normalerweise eine „1“ im Bit 2 des Steuerregisters gesetzt bekommen, damit auf das IOR, das Ein/Ausgaberegister, direkt zugegriffen werden kann.

Ein typischer Programmablauf wäre:

```
LDA #$ØF       „00001111“ =
               4 EINGABEKANÄLE,
               4 AUSGABEKANÄLE
STA DDRA       FESTLEGUNG
               DATENRICHTUNG
LDA #STEUER    STEUERFUNKTIONEN:
               BIT 2 = „1“ DAMIT DIREKT-
               ZUGRIFF AUF IORA
STA CRA        FESTLEGUNG
               STEUERFUNKTIONEN
```

Ein/Ausgabe

Eine Datenausgabe über Tor A würde mit den folgenden zwei Instruktionen durchgeführt (unter der Voraussetzung, daß das CRA-Bit 2 = „1“ ist):

```
LDA #DATEN     DATENWORT LADEN
STA IORA       DATENAUSGABE
```

bzw. mit Abruf aus Speicher S20:

```
LDA $20        DATENWORT LADEN
STA IORA       DATENAUSGABE
```

Das Einlesen von an den 6520 angelegten Daten sieht folgendermaßen aus:

```
LDA  IORA     DATENWORT EINLESEN
STA  $20      IN SPEICHER RETTEN
```

Wir retten hier den Inhalt des Akkumulators sofort in den Speicher 20 (hexadezimal). Jedoch ist diese Zeile nicht immer nötig. In vielen Fällen werden wir den Inhalt von IORA lediglich in den Akkumulator einlesen und dann vielleicht seinen Wert überprüfen, ihn jedoch nicht abspeichern.

6520 Warnungen

Zusätzlich zu den Unterschieden zwischen Tor A und Tor B sollten einige Besonderheiten der Steuerfunktionen in Erinnerung gerufen werden. Insbesondere sind die Bits 6 und 7 von A oder B zurückgesetzt, wenn 6 ein Eingabekanal ist und gelesen wird. Weiter *liest* man Daten von Tor B, um Bit 7 zurückzusetzen. Die Leitung CB2 für Quittungsbetrieb ist im Gegensatz zu CA2 nur zur *Ausgabe* von Daten über Tor B geeignet (CA2 arbeitet für *Eingabe und für Ausgabe*). Schließlich beachtet man, daß Bit 6 oder 7 einen Interrupt auslösen kann.

Abfragen mehrerer 6520-Einheiten

Der einfachste Weg, mehrere 6520 abzufragen, ist die Abfrage der Statusbits 6 und 7 der Steuerregister. Wenn beide Bits „0" sind, liegt bei dieser Einheit nichts vor. Wenn eines der beiden Bits „1" ist, wurde ein interner Interrupt erzeugt und muß abgearbeitet werden.

Methode 1

Um schnell festzustellen, welcher von vier Bausteinen eine Bedienung durch das System angefordert hat, kann man die Methode des sequentiellen Tabellenzugriffs anwenden. Voraussetzung hierfür ist, daß die Adressen der vier Einheiten hintereinander im Speicher abgelegt sind. Die Adresse n wird CRA1 zugewiesen sein, die Adresse n+1 CRB1, die Adresse n+2 CRA2, die Adresse n+3 CRB2, etc. Das Programm kann dann die Methode der indizierten indirekten Adressierung anwenden. Es kann folgendermaßen aussehen:

```
START     LDX  #8          ZÄHLER
NEXT      LDA  (TAB-1,X)   ZUGRIFF NÄCHSTES
                           STEUERREGISTER
          BMI  SERVICE     INTERRUPT-
                           ANFORDERUNG?
          DEX              X = X − 1
          BEQ  START
          BNE  NEXT        SCHLEIFE
```

```
TAB    -WORT CRA1 PIO #1   TOR A
       -WORT CRB1          TOR B
       -WORT CRA2 PIO #2   TOR A
       -WORT CRB2          TOR B
       -WORT CRA3 PIO #3   TOR A
       -WORT CRB3          TOR B
       -WORT CRA4 PIO #4   TOR A
       -WORT CRB4          TOR B
```

Bild 2.12: Identifizierung des PIO

Der Zähler, das Indexregister X, wird mit dem Startwert „8“ geladen und bei jedem Durchlaufen der Abfrageschleife um 1 erniedrigt (dekrementiert). Der Akkumulator wird zuerst mit dem Inhalt der letzten Tabellenadresse geladen:

```
LDA  (TAB-1,X)
```

Falls Bit 7 gesetzt war (Bit 7 ist das Vorzeichen- oder N-Flag), führt das Programm einen Sprung zu der Routine mit dem Namen „SERVICE“ durch, in der Interrupt dann behandelt wird:

```
BMI  SERVICE
```

Falls das N-Flag nicht gesetzt war, wird X dekrementiert und das nächste Steuerregister wird abgefragt:

```
DEX
BEQ  START      NEUER ZYKLUS BEI X = 0
BNE  NEXT       SCHLEIFE BEI X/0
```

Verbesserungsvorschlag: Würde ein Vertauschen der beiden letzten Programmanweisungen das Programm schneller machen?

Methode 2

In jedem Steuerregister müssen zwei Statusbits überprüft werden: die Bits 6 und 7. Der 6502-Befehl „BIT“ wurde extra für diesen Fall geschaffen. Es handelt sich um einen nichtlöschenden Vergleichsvorgang, mit dem man die Inhalte der Bits 6 und 7 testen kann. Das Programm für eine solche Abfrageschleife für mehrere 6520 zeigt Bild 2.13.

```
          BIT  CRA
          BMI  IRQA7
          BVC  NEXT1
IRQA6     ...            A2 IRQ GEFUNDEN (BIT 6)
.
.
.
```

```
IRQA7       ...         A1 IRQ GEFUNDEN (BIT 7)
.
.
.
NEXT1       BIT   CRB   DASSELBE FÜR TOR B
            BMI   IRQB7
            BVC   NEXT2
IRQB6       ...         B2 IRQ GEFUNDEN (BIT 6)
.
.
.
IRQB7       ...         B1 IRQ GEFUNDEN (BIT 7)
.
.
.
NEXT2       BIT   ...   NÄCHSTE 6520-EINHEIT
            .
            .
```

Bild 2.13: Identifizierung der Tore

Die „BIT"-Anweisung wird hier verwendet, um zu testen, ob Bit 6 oder Bit 7 eine „1" enthält. Dieses erledigt die Zeile:

```
BIT   CRA
```

Wir müssen dann testen, ob Bit 6 oder Bit 7 des CRA gesetzt war. Die Bit-Anweisung setzt die Flags V und N, so daß wir diese Flags jetzt testen können:

```
BMI   IRQA7     BIT 7 = „1"?
BVC   NEXT1     KEINE INTERRUPT-
                ANFORDERUNG
```

Falls keines der beiden Flags gesetzt war, erfolgt ein Sprung zu NEXT1, wo dann das CRB getestet wird. Bit 7 wird durch den Befehl BMI getestet. Falls Bit 7 gesetzt war, hat der Bit-Befehl das Negativ-Flag gesetzt und die Routine unter der Adresse IRQA7 wird abgearbeitet.

Andernfalls war Bit 6 das gesetzte Bit und die Routine unter der Adresse IRQA6, die hinter dem BVC-Befehl steht, wird abgearbeitet.

Diese Programmfolge kann für eine beliebige Anzahl von 6520's angewendet werden. Beachten Sie, daß diese Prozedur dem Bit 7 eine höhere Priorität als dem Bit 6 zuweist.

Der 6522

Der 6522, der von MOS Technology eingeführt wurde und auch von Rockwell International und Synertek hergestellt wird, ist eine verbesserte Version des 6520-Bausteins.

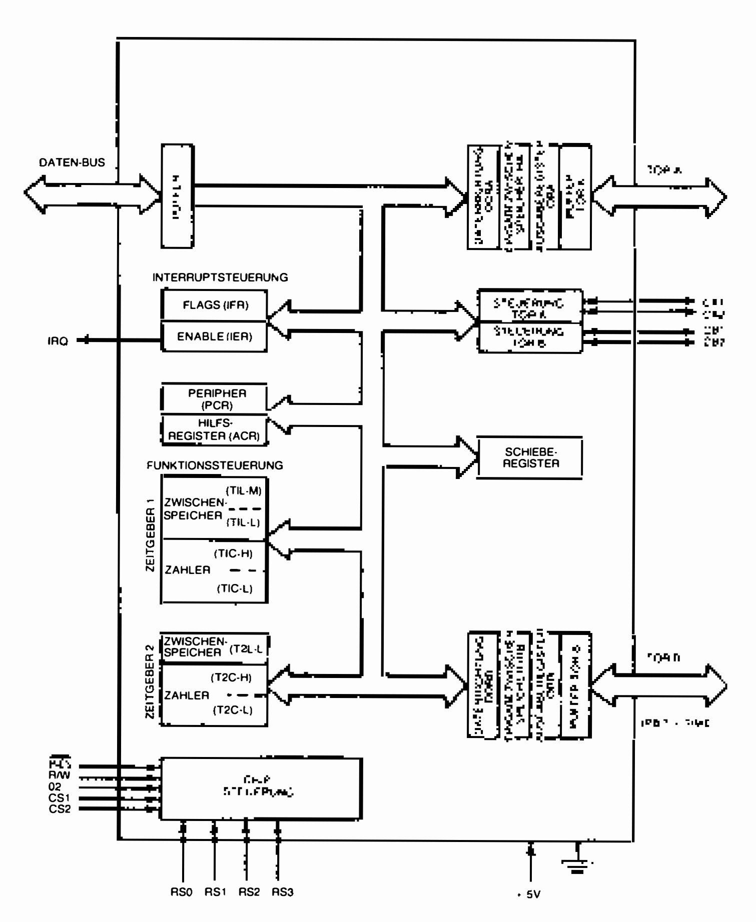


Bild 2.14: 6522 interne Architektur

Der 6522-Chip wird „versatile interface adapter" oder kurz „VIA" genannt, was etwa soviel wie „vielseitiger Interfaceadapter" heißt, und ist eine PIO-Zeitgeber-Schieberegister-Kombination. Er ist mit 16 internen Registern ausgerüstet, die Sie in Bild 2.14 sehen. Das entsprechende Registerverzeichnis zeigt Bild 2.15.

Es lassen sich vier Registerblöcke nach ihrer Funktion unterscheiden:

1. Die PIO Register (Adressen 0 bis 3 und Adresse F).
2. Die Zeitgeberregister (zwei Zeitgeber, Adressen 4 bis 9).
3. Das Schieberegister (Adresse A).
4. Die Steuerregister (Adressen B bis E).

Diese vier Registerblöcke werden wir jetzt im Detail untersuchen, um die Möglichkeiten des 6522 kennenzulernen.

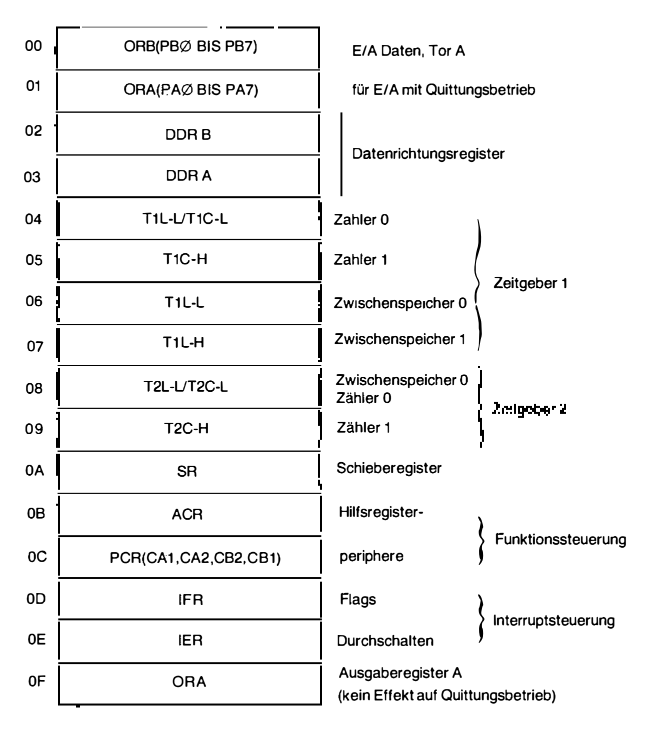


Bild 2.15: 6522 VIA Registerverzeichnis

RS3	RS2	RS1	RS0		Register		
0	0	0	0	00	ORB		Prallel E/A
0	0	0	1	01	ORA	+ Quittungsbetrieb	
0	0	1	0	02	DDRB		
0	0	1	1	03	DDRA		
0	1	0	0	04	T1L-L(W)/T1C-L(R)	+ löscht T1 Int Flag (R)	Zeitgeber 1
0	1	0	1	05	T1C-H(R)/T1L-H+T1C-H(W)	+ T1C-L/T1L-L + löscht T1 Int Flag (R)	
0	1	1	0	06	T1L-L		
0	1	1	1	07	T1L-H	+ löscht T1 Int Flag (W)	
1	0	0	0	08	T2L-L(W)/T2C-L(R)	löscht T2 Int Flag (W)	Zeitgeber 2
1	0	0	1	09	T2C-H	+ T2C-L/T2L-L löscht T2 Int Flag (W)	
1	0	1	0	0A	SR		Schieberregister
1	0	1	1	0B	ACR		Steuerung
1	1	0	0	0C	PCR		
1	1	0	1	0D	IFR		
1	1	1	0	0E	IER		
1	1	1	1	0F	ORA	[illegible]	[illegible]

Bild 2.16: 6522 Register

Der PIO-Teil

Der PIO-Teil erzeugt zwei bidirektionale 8-Bit Tore. Jedes Tor ist mit einem Ein/Ausgaberegister ausgestattet. Diese werden ORA für Tor A und ORB für Tor B genannt. Sie sind in Bild 2.14 gezeigt. Jedes Register ist mit einem Richtungsregister gekoppelt, das DDRA bzw. DDRB heißt. Wenn ein Bit des Datenrichtungsregisters auf „1" gesetzt ist, so wird die entsprechende Leitung des OR als *Ausgabe*leitung interpretiert. Ist das Datenrichtungsbit „0", so ist die entsprechende Leitung ein *Eingabe*kanal. Diese Vereinbarung wurde getroffen, damit alle Leitungen nach Anlegen eines „reset" Eingabekanäle sind.

Es gibt in diesem PIO eine Asymmetrie: das Tor A ist mit zwei OR's ausgerüstet. Eins mit und eins ohne die Möglichkeit zum Quittungsbetrieb.

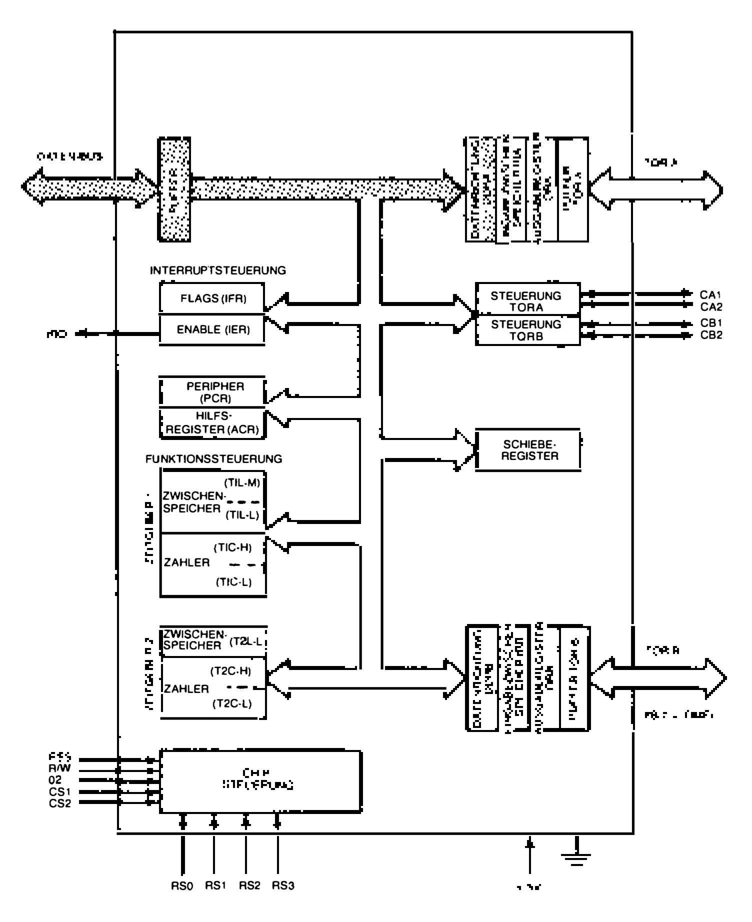


Bild 2.17: Gebrauch des 6522: STA DDRA

Der Gebrauch des PIO

Bevor man den PIO für Ein/Ausgabeoperationen verwendet, müssen erst die Datenrichtungsregister mit den richtigen Werten geladen werden, um die entsprechenden Bits der E/A-Register als Eingabekanäle oder Ausgabekanäle festzulegen. Als Beispiel wollen wir nun Tor A als Ausgabetor und Tor B als Eingabetor vereinbaren.

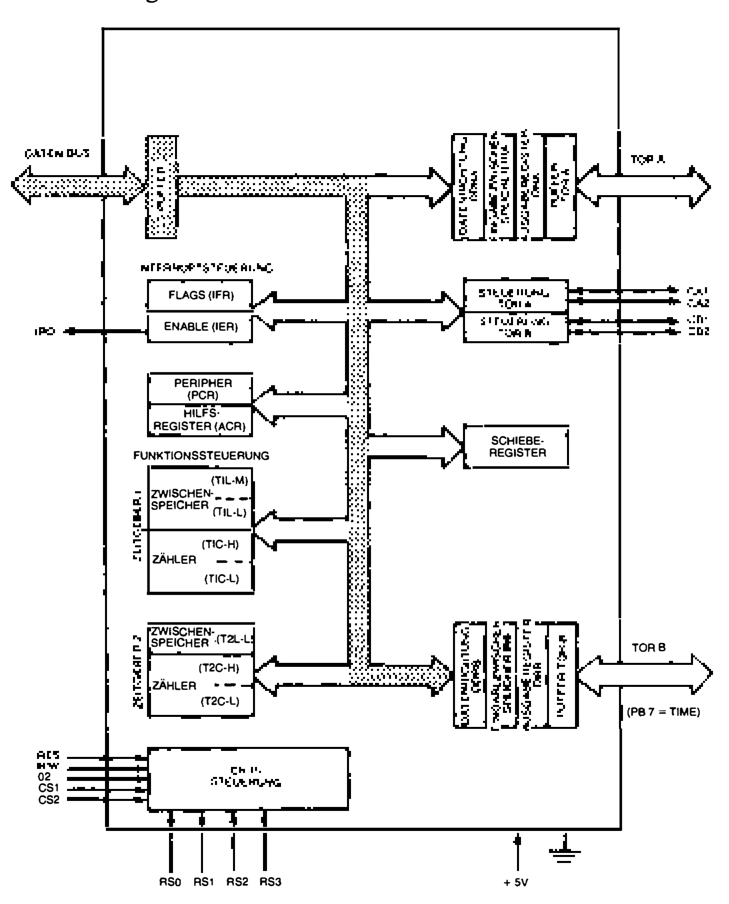


Bild 2.18: Gebrauch des 6522: STA DDRB

```
LDA  #$FF     „11111111“ = AUSGABE
STA  DDRA     A IST AUSGABETOR
LDA  #$00     „00000000“ = EINGABE
STA  DDRB     B IST EINGABETOR
```

(beachten Sie hierzu auch Bild 2.17 und 2.18)

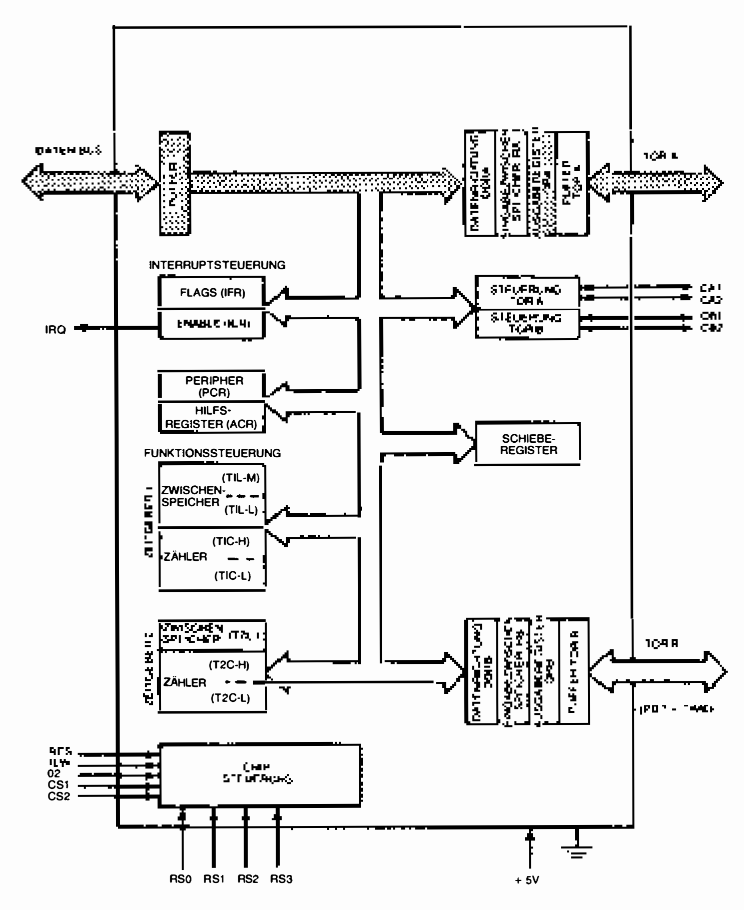


Bild 2.19: Gebrauch des 6522: STA ORA

Wir wollen nun über Tor A den Wert „00000001" ausgeben (siehe Bild 2.19):

```
LDA  #$01        „00000001"
STA  ORA
```

Zum Schluß wollen wir noch den am Tor B anliegenden Wert in den Akkumulator einlesen (Bild 2.20):

```
LDA  ORB
```

Normalerweise ist es bei jedem Zugriff auf die OR-Register nötig, ein *Status*signal zu testen, um sicherzugehen, daß die angesprochene Einheit für eine Ein- oder Ausgabe auch *bereit* ist. Man nennt das *Quittungsbetrieb*

oder *handshaking*. Die Funktion der Steuersignale, die benötigt werden, um dies auszuführen, werden wir jetzt erklären.

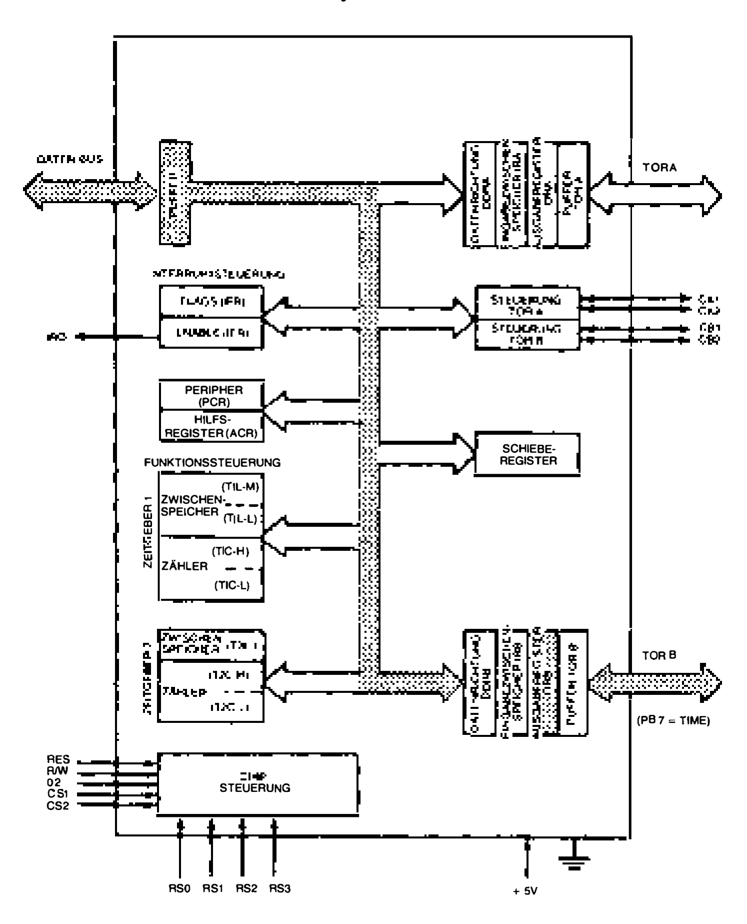


Bild 2.20: Gebrauch des 6522: LDA ORB

Die zwei Steuerleitungen (peripheres Steuerregister)

Jedes Tor ist mit zwei Steuerleitungen ausgerüstet: CA1, CA2 und CB1, CB2 (Bild 2.14, rechts). Beispielsweise muß der Mikroprozessor, bevor er Daten an einen Drucker sendet, etwa an einen Fernschreiber, erst die Gewißheit haben, daß der Drucker im Moment nicht beschäftigt ist, sondern daß er bereit ist, das nächste Zeichen zu empfangen. Dies wird durch das Verfahren des *Quittungsbetriebs* erreicht.

Wenn der Drucker nicht mehr *beschäftigt* ist, ist er *bereit*, das nächste Zeichen anzunehmen, und er wird einen Impuls oder ein Stufensignal zum

6522 schicken. Dieses Signal muß vom Baustein erkannt und zwischengespeichert und dann vom Programm abgefragt werden. Das Signal wird an einen der Steuereingänge CA1 oder CB1 gesendet.

Der 6522 erlaubt bei der Festlegung, wie das ein- oder ausgehende Signal beschaffen sein soll, ein Höchstmaß an Flexibilität.

Es ist möglich, festzulegen, ob ein *1/0-Übergang* (*negative* Flanke, Abwärtsstufe) oder ein *0/1-Übergang* (*positive* Flanke, Aufwärtsstufe) das interne Interruptflag setzen soll. Dies wird durch Bit 0 (für CA1) und Bit 4 (für CB1) des *peripheren Steuerregisters* (PCR) festgelegt. Eine „0" entspricht dem 1/0-Übergang, eine „1" dem 0/1-Übergang (Bild 2.21).

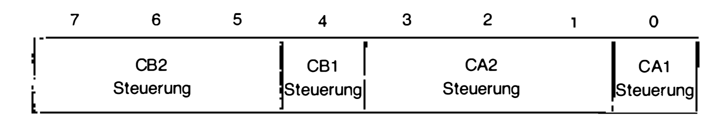


Bild 2.21: Peripheres Steuerregister

7	6	5	4	3	2	1	0
IRQ(R) EN(W)	T1	T2	CB1	CB2	SR	CA1	CA2

Bild 2.22: Interrupt Flag/Enable Register (IFR/IER)

CR BIT 1	0	Aktive Flanke des Eingangssignales	IRQ Ausgabe
0	0	negativ	nicht zugelassen (disable), „1"
0	1	negativ	zugelassen (enable), geht auf „0", wenn CRA Bit7 durch CA1/CB1-Übergang gesetzt wird.
1	0	positiv	nicht zugelassen (disable), „1"
1	1	positiv	zugelassen (enable) (wie oben)

Bild 2.23: Funktionen der Steuerleitungen (ACR)

Wenn die Art des Signals einmal festgelegt ist, wird es möglich, es zu testen.

Testen des Status: Es ist möglich, festzustellen, ob ein Übergang stattgefunden hat, indem man die Inhalte der Bits 1 und 4 des *Interrupt Flag Registers* (IFR) (Bild 2.22) testet (Bit 1: CA1, Bit 4: CB1). Diese Bits werden „0" sein, solange kein Signal empfangen worden ist, und werden auf „1" gesetzt sein, sobald der entsprechende Übergang aufgetreten ist. Nachdem ein „1"-Zustand gefunden wurde, muß es möglich sein, diesen *zurückzusetzen,* so daß man zum Nachweis des nächsten Ereignisses übergehen kann. Man bewerkstelligt dies entweder, indem man das entsprechende Bit des Registers mit einer „1" überschreibt, oder aber durch Lesen oder Schreiben des entsprechenden Ein/Ausgaberegisters.

PCR3	PCR2	PCR1	Modus
0	0	0	CA2 Interruptmodus, negativ flankengetriggert (IFRO/ORA zurückgesetzt) – setzt CA2 Interruptflag (IFRO) bei negativem Sprung des Eingangssignals. Löschen des IFRO durch Lesen oder Schreiben des peripheren Ausgaberegisters A (ORA) oder durch Laden von logisch 1 in das IFRO.
0	0	1	CA2 Interruptmodus, negativ flankengetriggert (IFRO zurückgesetzt) – setzt IFRO bei negativem Sprung des CA2 Eingangssignals. Lesen oder Schreiben des ORA löscht das CA2 Interruptflag nicht. Löschen des IFRO durch Laden von logisch 1 in das IFRO.
0	1	0	CA2 Interruptmodus, positiv flankengetriggert (IFRO/ORA zurückgesetzt) – setzt CA2 Interruptflag bei positivem Sprung des CA2 Eingangssignals. Löschen des IFRO durch Lesen oder Schreiben des peripheren Ausgaberegisters A.
0	1	1	CA2 Interruptmodus, positiv flankengetriggert (IFRO zurückgesetzt) – setzt IFRO bei positivem Sprung des CA2 Eingangssignals. Lesen oder Schreiben des ORA löscht das CA2 Interruptflag nicht. Löschen des IFRO durch Laden von logisch 1 in das IFRO.
1	0	0	CA2 Quittungsbetrieb-Ausgabemodus – setzt Ausgang CA2 auf „0" beim Lesen oder Schreiben des peripheren Ausgaberegisters A. Zurücksetzen von CA2 auf „1" durch Signal an CA1.
1	0	1	CA2 Pulsausgabemodus – CA2 geht nach Lesen oder Schreiben des peripheren Ausgaberegisters A für einen Taktzyklus auf „0".
1	1	0	CA2-Ausgang, σ-Modus – Der CA2-Ausgang bleibt in diesem Modus auf „0" (low).
1	1	1	CA2-Ausgang, 1-Modus – Der CA2-Ausgang bleibt in diesem Modus auf „1" (high).

Bild 2.24: Funktionen des PCR im Detail (Quelle: Rockwell)

PCR7	PCR6	PCR5	Modus
0	0	0	CB2 Interruptmodus, negativ flankengetriggert (IFR3/ORB zurückgesetzt) – setzt CB2 Interruptflag (IFR3) bei negativem Sprung des CB2 Eingangssignals. Löschen des IFR3 durch Lesen oder Schreiben des peripheren Ausgaberegisters B (ORB) oder durch Laden von logisch 1 in das IFR3.
0	0	1	CB2 Interruptmodus, negativ flankengetriggert (IFR3 zurückgesetzt) – setzt IFR3 bei negativem Sprung des CB2 Eingangssignals. Lesen oder Schreiben des ORB löscht das Interruptflag nicht. Löschen des IFR3 durch Laden von logisch 1 in das IFR3.
0	1	0	CB2 Interruptmodus, positiv flankengetriggert (IFR3/ORB zurückgesetzt) – setzt CB2 Interruptflag bei positivem Sprung des CB2 Eingangssignals. Löschen des CB2 Interruptflags durch Lesen oder Schreiben des ORB oder durch Laden von logisch 1 in das IFR3.
0	1	1	CB2 Interruptmodus, positiv flankengetriggert (IFR3 zurückgesetzt) – setzt IFR3 bei positivem Sprung des CB2 Eingangssignals. Lesen oder Schreiben des ORB löscht das CB2 Interruptflag nicht. Löschen des IFR3 durch Laden von logisch 1 in das IFR3.
1	0	0	CB2 Quittungsbetrieb-Ausgabemodus – setzt CB2 auf „0“ durch Schreiben des ORB. Zurücksetzen von CB2 auf „1“ durch CB1 Eingangssignal.
1	0	1	CB2 Pulsausgabemodus – CB2 geht nach Schreiben des peripheren Ausgaberegisters B für einen Taktzyklus auf „0“.
1	1	0	CB2 manueller „0“-Ausgabemodus – der CB2 Ausgang wird in diesem Modus auf „0“ gehalten.
1	1	1	CB2 manueller „1“-Ausgabemodus – der CB2 Ausgang wird in diesem Modus auf „1“ gehalten.

Bild 2.25: Funktionen des PCR im Detail

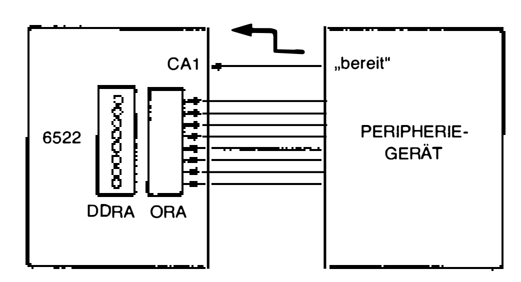


Bild 2.26: Einlesen von Daten im Bereit-Zustand des Gerätes

Ein einfaches Eingabespiel

Wir wollen als Bereitschaftssignal von der Peripherie einen 0/1-Übergang (positive Flanke) und Tor A als Eingabetor vereinbaren (Bild 2.26). Stets, wenn Daten bereitstehen, werden sie in den Akkumulator eingelesen. Das Programm lautet:

```
                 LDA  #$00
                 STA  DDRA      TOR A = EINGABETOR
                 LDA  #$01
                 STA  PCR       CA1 INTERRUPT DURCH
                                POSITIVE FLANKE
WARTE            LDA  IFR       LIES INTERNE FLAGS
                 AND  #$02      00000010 MASKIERE BIT 1
                                FÜR CA1
                 BEQ  WARTE     BEREIT?
                 LDA  ORA       LIES DATEN EIN
```

Verbesserung: Können Sie die zwei Befehle ‚LDA IFR' und ‚AND #$02' so abändern, daß die Leistungsfähigkeit verbessert wird?

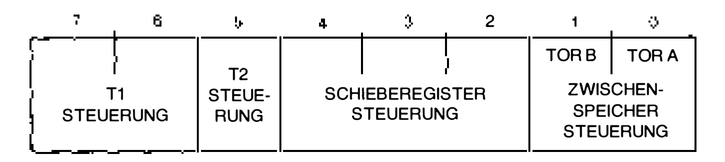


Bild 2.27: Hilfs-Steuerregister (ACR)

Zwischenspeicherung von Ein/Ausgaben

Eingabe und Ausgabe des 6522 sind nicht symmetrisch. Ausgaben werden stets *zwischengespeichert*. Das ist der Grund, warum das Ein/Ausgaberegister OR (output-register, d. h. *Ausgabe*register) genannt wird. *Eingaben werden nicht unbedingt zwischengespeichert.* Das wird durch die Bits 0 und 1 (0 für Tor A und 1 für Tor B) des Hilfs-Steuerregisters (auxiliary control register, kurz ACR) festgelegt. Wenn diese Bits „0" sind, wird bei Eingaben nicht zwischengespeichert. Wenn diese Bits auf „1" gesetzt sind, werden die Eingaben zwischengespeichert (Bild 2.27). Wenn eine Eingabe nicht zwischengespeichert wird, dann liest das Programm den aktuellen Wert der Eingabekanäle, die an das Tor angeschlossen sind, das gerade gelesen wird. Wenn die Eingaben zwischengespeichert werden, wird der Zwischenspeicher durch das Signal am Eingang CA1 oder CB1 aktiviert, je nachdem, welches Tor verwendet wird. Dieser Wert wird dann im Zwischenspeicher solange gehalten, bis auf der Steuerleitung der nächste Impuls empfangen wird. *Vorsicht:* bei der Vereinba-

rung von *Ausgabe*kanälen liest das Programm das Zwischenspeicher-Steuerregister, das denselben oder einen anderen Inhalt wie das OR haben kann.

Ausgabe eines Steuersignales

CA2 oder CB2 werden verwendet, um Steuersignale zu erzeugen (Bild 2.14). Da diese Leitungen bidirektional sind, muß die Ausgabefunktion durch Setzen der Bits 3 oder 7 (für CA2 oder CB2) des peripheren Steuerregisters vereinbart werden (Bild 2.24).

Das Signal kann entweder als Stufenimpuls oder als Impuls festgelegt werden. Eine „0" in den Bits 2 oder 6 (für A oder B) entspricht einem *Impuls*. Eine „1" entspricht einem *Stufen*impuls. Wurde ein Stufenimpuls vereinbart, ist es möglich, eine positive oder eine negative Flanke festzulegen. Man erreicht dies durch Setzen oder Rücksetzen von Bit 1 oder 5 (für A oder B) (Bild 2.24).

Schließlich kann, wenn ein Impuls erzeugt wird, dessen Länge mit Bit 1 und 5 (jeweils für CA2 und CB2) des Steuerregisters beeinfluß werden. Ist dieses Bit auf „0" gesetzt, wird ein Einzel-Takt-Impuls erzeugt. Ist es auf „1" gesetzt, wird am Ausgang ein Impuls erzeugt, der von dem Zeitpunkt des Zugriffs auf das OR (Lesen oder Schreiben) bis zum nächsten Signal an CA1 bzw. CB1 auf „0" bleibt.

Zusammenfassung: Ausgabe von Steuersignalen

Ein Impuls von praktisch jeder Dauer und Polarität kann vereinbart werden. Er kann dazu verwendet werden, externe Geräte abzufragen, einen Datentransfer mit einem anderen Gerät zu bestätigen, das an derselben Leitung angeschlossen ist oder den Zustand solcher Geräte zu steuern (ein, aus oder andere Optionen).

Eine Zusammenfassung der Bits des peripheren Steuerregisters ist in Bild 2.21 gezeigt, Details entnehmen Sie den Bildern 2.24 und 2.25.

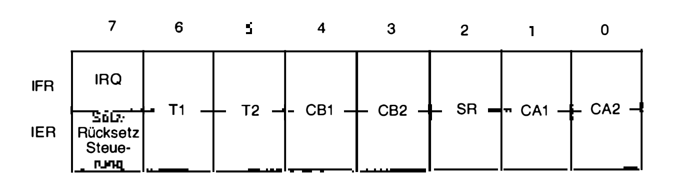


Bild 2.28: Die Interrupt-Register (IFR/IER)

Interrupts

Interrupts (Programmunterbrechungen) werden von zwei Registern gesteuert, dem Interrupt Enable Register (IER) und dem Interrupt Flag Register (IFR). Bild 2.28 zeigt diese Register. Sie belegen dieselbe Speicheradresse. Das ist ein Eingaberegister, das andere ein Ausgaberegister.

Das Interrupt Flag Register IFR ist ein Eingaberegister. Jedes Bit von 0 bis 6 wird gesetzt, wenn auf irgendeiner der externen Leitungen (CA1, CA2, CB1, CB2), im Schieberegister (SR) oder einem der beiden Zeitgeber (T1 und T2) ein Interrupt ausgelöst wird. Bit 7 wird stets dann gesetzt, wenn irgendeines der anderen Bits des Registers gesetzt ist.

Das Interrupt Enable Register IER wird Interrupts von igendeiner dieser Quellen zulassen (enable) oder sperren (disable). Die Bedeutungen der Bits im IER stimmen mit denen im IFR überein (Bild 2.28). Wenn eines der Bits auf „0“ gesetzt ist, ist der entsprechende Interrupt *gesperrt* und wird nicht registriert. Wenn es auf „1“ gesetzt ist, ist er *zugelassen* und wenn ein Interrupt auftritt, wird er registriert. Das Programm kann dann den Inhalt des IFR Registers lesen und jedes wichtige Bit überprüfen, um festzustellen, ob ein Interrupt vorliegt. Um jedes IER-Bit bequem setzen oder löschen zu können, wird Bit 7 des IER in Verbindung mit einem Lese- oder Schreibsignal verwendet, und der Inhalt des Datenbusses wird dann in das IER kopiert. Wenn das Bit 7 „0“ ist, wird jede „1“ das entsprechende enable-Flag löschen. Wenn das Bit 7 „1“ ist, wird jede in das IER geschriebene „1“ das entsprechende enable-Flag setzen.

Beispiel: Wir wollen Interrupts von CA1 und CA2 zulassen, alle anderen wollen wir sperren (Bild 2.28):

```
LDA  #$7C        „01111100“ =
                 LÖSCHE BITS 2 BIS 6
STA  IER
LDA  #$83        „10000011“ =
                 SETZE BITS 0 UND 1
STA  IER
```

Übungsaufgabe 2.1: Schreiben Sie ein Programm, das CB1 Interrupts zuläßt und alle anderen sperrt.

Übungsaufgabe 2.2: Sperren Sie CB1 und CB2, ohne die anderen zu verändern.

Interrupt-Erkennung

Falls mehrere Interrupts gleichzeitig auftreten können, d. h. falls mehrere Bits des IFR verwendet werden, muß das Programm den Inhalt des IFR testen und feststellen, welcher Interrupt aufgetreten ist. Die Reihenfolge, mit der diese Bits getestet werden, legt die Priorität der entsprechenden Interrupts fest. Hat beispielsweise der Interrupt von T1 die höchste Priorität, dann muß dieses Bit als erstes getestet werden. Die einfachste Methode, den Inhalt des IFR zu überprüfen, ist das Durchschieben seines In-

halts von rechts oder links, und durch Überprüfung des Übertrag-Flags (carry) dasjenige Bit zu testen, das in das Übertragbit geschoben wurde. Diese Technik weist den Bits in Bild 2.28 die Priorität in der Reihenfolge von links nach rechts oder von rechts nach links zu.

Übungsaufgabe 2.3: Sehen Sie sich Bild 2.28 an. Sortieren Sie die möglichen Interrupts nach ihrer effektiven Priorität. Nehmen Sie dabei an, daß der Inhalt des IFR von dem Abfrageprogramm nach links durchgeschoben wird.

Natürlich ist es auch möglich, Kombinationen einzelner Interrupts zu testen, indem man nur die Werte der betreffenden Bits im IFR testet. Für weitere Details über Interrupts und Abfragemethoden lesen Sie bitte das Kapitel 3 des Buchs *„Programmierung des 6502"*.

Die Zeitgeber

Der 6522 ist mit zwei *Intervall-Zeitgebern* ausgerüstet. Diese Zeitgeber können für Eingabe- und Ausgabeoperationen verwendet werden.

Bei der Verwendung für *Ausgabe* kann der Zeitgeber Einzelimpulse oder Impulsketten erzeugen.

Bei Verwendung für *Eingabe* mißt der Zeitgeber entweder die Länge eines Impulses oder zählt die Anzahl eingehender Impulse. Wenn der Zeitgeber nur *einen* Impuls bestimmter Dauer erzeugt oder empfäng, spricht man von *„Einzelbetrieb"*. In dieser Betriebsart kann sowohl Zeitgeber 1 als auch Zeitgeber 2 des 6522 betrieben werden.

Wenn man den Zeitgeber zur Erzeugung oder zum Abzählen von Impulsketten benutzt, spricht man vom *„Freilaufmodus"*. In dieser Betriebsart kann nur der Zeitgeber 1 verwendet werden.

Bevor man einen der beiden Zeitgeber für Ausgabeoperationen verwendet, muß sein Zählregister mit einem Wert geladen werden: bei der Erzeugung von Impulsen wird der Zähler entweder die Anzahl der zu erzeugenden Taktimpulse oder aber die Dauer der Impulse enthalten.

Bei Verwendung des Zeitgebers für Eingabeoperationen muß sein Zählregister gelöscht werden. Wenn er Impulse zählt, wird dieses Register die Anzahl bis dahin eingegangener Impulse enthalten. Beim Messen von Einzelimpulsen wird es die Impulsdauer enthalten.

Zeitgeber 1 gegenüber Zeitgeber 2

Zeitgeber 2 kann bei der Eingabe verwendet werden, um Impulse zu zählen, die an PB6 des IORB anliegen (Bild 2.14). Wird er für die Ausgabe verwendet, kann er jedoch nur ein einzelnes Ausgangssignal von vorgegebener Dauer an PB6 ausgeben. Er kann keine Impulsketten erzeugen. Einer dieser zwei Betriebszustände wird durch Bit 5 des Hilfs-Steuerregisters (ACR) ausgewählt (Bild 2.27). „0" entspricht hierbei dem Einzelbetrieb, „1" entspricht dem Freilaufmodus (Impuls-Abzählung).

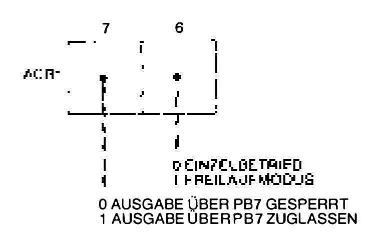


Bild 2.29: 6522: Hilfs-Steuerregister steuert Betriebszustände von Zeitgeber 1 (T1)

Zeitgeber 1 ist verschieden von Zeitgeber 2 und bietet zusätzliche Möglichkeiten. Er kennt vier verschiedene Betriebszustände, die in Bild 2.29 gezeigt sind. Er kann entweder im Einzelbetrieb oder im Freilaufmodus betrieben werden. Zusätzlich kann eine Ausgabe über PB7 entweder zugelassen oder gesperrt werden. Der Betriebszustand wird durch Bit 6 des Hilfs-Steuerregisters festgelegt. Es ist „0“ für Einzelbetrieb und „1“ für freilaufenden Betrieb.

Bit 7 legt fest, ob die Ausgabe über PB7 zugelassen oder gesperrt ist. Wenn Bit 7 „0“ ist, ist PB7 gesperrt, ist es „1“, so ist PB7 zugelassen (Bild 2.30).

ACR7 STEUERUNG PB7	ACR 6 STEUERUNG EINZEL/FREI	MODUS
0	0 (EINZEL-BETRIEB	Erzeugt Interrupt nach Ablauf der Zeit, mit der T1 geladen wurde. PB7 gesperrt.
0	1 (FREILAUF-MODUS)	Erzeugt Dauer-Interrupt. PB7 gesperrt.
1	0 (EINZEL-(BETRIEB)	Erzeugjedesmal, wenn T1 geladen wird, einen Interrupt und Ausgangssignal über PB7 = Einzelbetrieb mit programmierbarer Impulsbreite.
1	1 (FREILAUF-MODUS)	Erzeugt Dauer-Interrupt und Rechtecksignal auf PB7.

Bild 2.30: 6522 – Hilfssteuerregister wählt Betriebszustände von Zeitgeber 1 aus

Laden der Zählregister

Jeder Zeitgeber verwendet einen 16-Bit-Zähler. Zuerst muß das niederwertige Byte (engl.: Low byte) geladen werden, dann das höherwertige

(engl.: high byte). Das Laden des höherwertigen Teils des Zählers löscht automatisch das Interruptflag des Zeitgebers und startet den Zählvorgang. Der Zeitgeber 1 ist außerdem mit einem echten 16-Bit Zwischenspeicher ausgestattet, Zeitgeber 2 hingegen nicht. Das befähigt den Zeitgeber 1, kontinuierlich im Freilaufmodus zu arbeiten. Der zwischengespeicherte Wert wird automatisch an den Zähler weitergegeben, wenn der Zähler Null erreicht. Beim Zeitgeber 1 können die Zwischenspeicher gelesen oder geladen werden, ohne daß dies den Zähler beeinflußt. Man verwendet dies zur Erzeugung von beliebig kompliziert geformten Ausgangssignalen.

Die Einzelheiten der Adressierung des Zeitgebers werden im Bild 2.31 gezeigt.

	ADRESSE	SCHREIBEN	LESEN
ZEITGEBER 1	--04	T1L-L	T1C-L/ + lösche T1 Int Flag
	--05	T1L-H→T1C-H T1L-L→T1C-L + lösche T1 Int Flag	T1C-H
	--06	T1L-L	T1L-L
	--07	T1L-H + lösche T1 Int Flag	T1L-H
ZEITGEBER 2	--08	T2L-L	T2C-L + lösche T2 Int Flag
	--09	T2C-H T2L-L→T2C-L + lösche T2 Int Flag	T2C-H

Bild 2.31: Adressierung der Zeitgeber

Erläuterungen: T1L-L = timer1 latch low, d. h. Zeitgeber1-Zwischenspeicher, niederwertiges Byte; T1L-H = timer1 latch high, d. h. Zeitgeber1-Zwischenspeicher, höherwertiges Byte; T1C-L und T1C-H dasselbe mit „C" = counter, d. h. Zähler; Int Flag = Interrupt-Flag

Tatsächliche Impulsdauer

Die tatsächliche Form des Ausgangssignals von Zeitgeber 1 ist im Bild 2.32 gezeigt. Beachten Sie, daß die tatsächliche Dauer des Impulses entweder der gezählte Wert („N") plus 2 ist, oder der gezählte Wert plus 1,5. Um eine exaktere Zeitgabe zu erreichen, sollte der Programmie-

rer daher die gewünschte Anzahl von Perioden minus 2 in das Zählerregister laden.

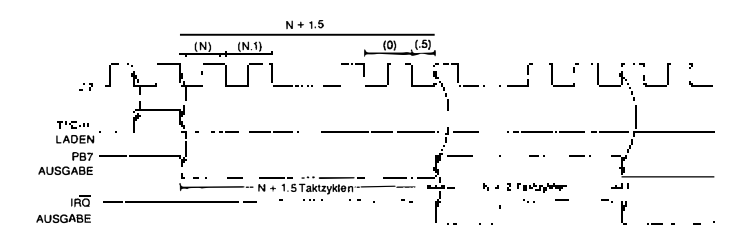


Bild 2.32: Zeitgeber 1 im Freilaufmodus

Das Schieberegister

Das Schieberegister ist für seriell-zu-parallel oder parallel-zu-seriell Umwandlung ausgelegt. Die Schiebe-Geschwindigkeit kann durch drei Quellen gesteuert werden: Zeitgeber 2, Phase 2 des Taktgenerators (Φ2) und einen externen Taktgenerator. Welche Quelle als Taktgeber verwendet wird, wird durch die Bits 2 und 3 des Hilfssteuerregisters festgelegt (Bild 2.27). Bit 4 des Hilfssteuerregisters legt fest, ob es sich um eine Eingabe oder um eine Ausgabe handelt. Bild 2.33 zeigt eine vollständige Tabelle, die die Funktionen dieser Bits erklärt.

ACR4	ACR3	ACR2	Modus
0	0	0	Schieberegister gesperrt.
0	0	1	Eingabe, Schieberegister gesteuert durch Zeitgeber 2.
0	1	0	Eingabe, Schieberegister gesteuert durch ∅2.
0	1	1	Eingabe, Schieberegister gesteuert durch externen Taktgenerator.
1	0	0	Ausgabe im Freilaufmodus, Frequenz durch Zeitgeber 2 bestimmt.
1	0	1	Ausgabe, Schieberegister gesteuert durch Zeitgeber 2.
1	1	0	Ausgabe, Schieberegister gesteuert durch ∅2.
1	1	1	Ausgabe, Schieberegister gesteuert durch externen Taktgenerator.

Bild 2.33: Steuerung des Schieberegisters

Bei Ausgabe wird der Programmierer das Schieberegister zuerst laden. Dadurch wird automatisch das Takten und Durchschieben gestartet. Sobald 8 Bits durch das Schieberegister durchgeschoben sind, wird automatisch das Interruptflag (Bit 2 des Interrupt Flag Registers) gesetzt. Dieses kann dann durch das Programm getestet werden.

Bei Eingabe muß das Schieberegister mit irgendeinem Wert initialisiert werden, z. B. mit dem Wert „0", um das Takten zu starten. Es wird dann beginnen, mit der durch die taktende Quelle vorgegebenen Frequenz Bits entgegenzunehmen. Die taktende Quelle ist Zeitgeber 2, Phase 2 des Taktgenerators, oder ein externer Taktgeber, so wie es die Bits 2,3,4 des ACR vorgeben. Sobald 8 Bits eingelesen sind, wird das entsprechende Flag des IFR gesetzt. Das Programm wird also irgendeinen Startwert wie etwa „0" in das SR laden und dann dauernd das Bit 2 des IFR testen. Sobald ein Interrupt gesetzt wird, ist der Eingabezyklus beendet, indem man die Bits 2,3,4 des ACR auf „0" zurücksetzt, während das Programm die Daten abspeichert. Wenn die Daten ohne Unterbrechung einlaufen, wird man das Schieberegister natürlich nicht sperren, und das Programm sollte schnell genug „zurück sein", um keine Daten zu verlieren.

Programmierung des 6522

Der 6522 ist eine Kombination von PIO, Zeitgeber und Schieberegister. Die Standardoperationen für Eingabe und Ausgabe mit dem PIO werden im wesentlichen genauso durchgeführt, wie mit dem 6520, mit der Ausnahme, daß die Register direkt angewählt werden, und daß man nicht das Bit 2 des Steuerregisters hin- und herschalten muß, um zwischen ihnen zu unterscheiden. Das führt zu einer einfacheren und gestraffteren Programmierung. Jedoch sind die Steuerungsmöglichkeiten durch den 6522 umfassender und völlig verschieden von denen des 6520. Wir wollen daher als erstes einige Beispiele für Standard-Ein/Ausgaben und im Anschluß einige Beispiele für Steuerungen studieren.

Standard-Eingabe

Die Eingabeoperation wird durchgeführt, indem man lauter Nullen in das Datenrichtungsregister desjenigen Tores lädt, das als Eingabetor dienen soll, und dann den Inhalt des OR liest. Wir werden in diesem einfachen Programm zusätzlich die eben eingelesenen Daten in den Speicher mit der Adresse $20 retten. Hier das Programm:

```
EINGABE    LDA  #$00     „00000000"
           STA  DDRA     TOR A IST EINGABETOR
           LDA  ORA      LIES DATEN EIN
                         (FALLS GÜLTIG)
           STA  $20      ABLEGEN IN SPEICHER
```

Standard-Ausgabe

Die Ausgabeoperation wird ganz genauso durchgeführt, wie die Eingabeoperation; das Datenrichtungsregister von Tor B wird mit lauter Einsen geladen, wodurch es zum Ausgabetor wird. Nehmen wir an, daß die Daten, die an das Tor B gegeben werden sollen, sich im Speicher $20 befinden. Das Datenwort wird also zuerst in den Akkumulator geladen und dann an das ORB weitergegeben. Der Leser wird sich erinnern, daß es beim 6502 keinen Befehl gibt, der einen direkten Datentransfer vom Speicher $20 zum ORB erlaubt. Es wird daher ein zusätzlicher Befehl benötigt, um die Daten zuerst vom Speicher in den Akkumulator, und dann vom Akkumulator zum ORB zu bringen. Das Programm lautet folgendermaßen:

```
AUSGABE    LDA #$FF     „11111111"
           STA DDRB     TOR B = AUSGABETOR
           LDA $20      HOLE DATEN
                         AUS SPEICHER
           STA ORB      DATENAUSGABE
```

HEX	RS3	RS2	RS1	RS0	L/S	REGISTER	BEMERKUNGEN
0	0	0	0	0	S	ORB	
0	0	0	0	0	L	IRB	
1	0	0	0	1	S	ORA	steuert Quittungsbetrieb
1	0	0	0	1	L	IRA	
2	0	0	1	0		DDRB	
3	0	0	1	1		DDRA	
4	0	1	0	0	S	T1L-L	Zwischenspeicher 1
4	0	1	0	0	L	T1C-L	Zähler 1
5	0	1	0	1	.	T1C-H	T1L-L nach T1C-L
6	0	1	1	0	.	T1L-L	
7	0	1	1	1		T1L-H	
8	1	0	0	0	S	T2L-L	Zwischenspeicher 2
8	1	0	0	0	L	T2C-L	Zähler 2
9	1	0	0	1		T2C-H	T2L-L nach T2C-L
A	1	0	1	0	.	SR	
B	1	0	1	1	.	ACR	
C	1	1	0	0	.	PCR	
D	1	1	0	1	.	IFR	
E	1	1	1	0		IER	
F	1	1	1	1	.	ORA	kein Einfluß auf Quittungsbetrieb

Bild 2.34: 6522 Registerwahl erfolgt direkt

Anwendung der Steuerfunktionen

Für diesen Abschnitt wollen wir vereinbaren, daß Tor A aus lauter Eingabekanälen besteht. Ferner wollen wir annehmen, daß das Peripheriegerät

oder die Schaltung, die an Tor A angeschlossen it, das „Daten bereit"-Signal auf die Leitung CA1 legt. Es soll hierfür ein 0/1-Übergang (positive Flanke) vereinbart sein. Der 6522 muß dieses „Daten bereit"-Signal erkennen und das Programm wird den 6522 abfragen, ob irgendwelche Daten empfangen worden sind. Falls Daten empfangen wurden, wird es diese lesen und in den Speicher mit der Adresse $20 ablegen. Das Programm wurde bereits entwickelt (siehe „Standard Eingabe", Seite 79). Wir führen es hier nochmals auf:

```
EINGABE     LDA  #$00     A = EINGABETOR
            STA  DDRA
            LDA  #$01     CA1 INT 0/1 ÜBERGANG
            STA  PCR
TEST        LDA  IFR      TESTE BIT 1
            AND  #$02     00000010 BINÄR, MASKE
            BEQ  TEST     = 0 ?
            LDA  ORA      LIES DATEN
            STA  $20      ABLEGEN IN SPEICHER
```

Wie üblich wird das Datenrichtungsregister mit lauter Nullen geladen, um ORA als Eingabetor zu vereinbaren:

```
LDA  #00
STA  DDRA
```

Nun wird im Steuerregister PCR vereinbart, daß ein interner Interrupt erzeugt wird, wenn ein 0/1-Übergang auftritt:

```
LDA  #01
STA  PCR
```

Die beiden obigen Befehle laden den binären Wert „00000001" in das PCR. Der Leser sollte sich anhand von Bild 2.21 überzeugen, daß dies tatsächlich der korrekte Wert ist. Das Bit Null des peripheren Steuerregisters PCR legt fest, welche Flanke des Eingangssignales erwartet wird. Da wir das CA1 Interruptflag durch eine positive Flanke triggern wollen (0/1-Übergang), muß also PCRO auf den Wert 1 gesetzt werden.

Die Bits 6 und 7 des Hilfssteuerregisters ACR beziehen sich auf den Betriebszustand des Zeitgebers 1. Da dieser Zeitgeber hier nicht verwendet wird, ist deren Inhalt im Moment unwichtig. Die Bits 2, 3 und 4 legen die Funktionen des Schieberegisters fest. Da das Schieberegister nicht gebraucht wird, sollten sie Null sein, wie man Bild 2.33 entnehmen kann. Bit 5 des ACR steuert den Zeitgeber 2 und ist daher unbenutzt. Bit 1 regelt die Zwischenspeicherung von Tor B und ist daher ebenfalls unbenutzt. Bit Null regelt die Zwischenspeicherung von Tor A.

Wenn Zwischenspeicherung (durch Setzen einer „1") vereinbart ist, werden Daten, die am Eingang von Tor A anliegen, stets dann zwischengespeichert, wenn ein „Daten bereit"-Signal an CA1 empfangen wird. Hierdurch wird auch der CA1-Interrupt ausgelöst. Wir setzen also ACRO auf „1":

```
        LDA  #01
        STA  ACR
```

Da wir hier annehmen, daß Abfrageverfahren anstelle von Hardware-Interrupts angewendet werden, wird das Programm dafür verantwortlich sein, daß der Inhalt des Interruptflags gelesen und festgestellt wird, ob eine Interruptanforderung vorliegt. Den Inhalt des Interrupt-Flagregisters zeigt Bild 2.28. Das Bit 1 des IFR muß getestet werden, um festzustellen, ob das „Daten bereit"-Signal von CA1 empfangen worden ist. Das wird durch die folgenden drei Instruktionen durchgeführt:

```
TEST    LDA  IFR
        AND  #$02
        BEQ  TEST
```

Der AND-Befehl unterdrückt alle Bits außer Bit 1, so daß dieses getestet werden kann (Maskierung).

Solange Bit 1 Null ist, wird das Programm in dieser Abfrageschleife bleiben. Sobald das „Daten bereit"-Signal erkannt wird, können Daten aus dem ORA ausgelesen werden und zu ihrem endgültigen Speicherplatz transferiert werden. Wiederum wollen wir annehmen, daß dies der Speicherplatz $20 sei:

```
        LDA  ORA
        STA  $20
```

Das Lesen des Inhalts des ORA in den Akkumulator wird außerdem automatisch Bit 1 des IFR (den CA1 Status-Anzeiger) löschen, so daß der interne Interrupt automatisch zurückgesetzt wird.

Wenn Interrupts verwendet werden, müssen sie stets explizit zurückgesetzt werden. Es ist wichtig, das im Gedächtnis zu behalten. Der 6522 ist derart organisiert, daß der „normale" Programmablauf, wie etwa das lesen des ORA-Inhaltes nach dem Auftreten eines Interrupts, hierauf automatisch Rücksicht nimmt. Jedoch sollte sich der Leser der Tatsache bewußt sein, daß bei der Anwendung ausgefallenerer Programmiertechniken Fehler dadurch auftreten können, daß das Interruputflag dauernd gesetzt bleibt. Eine Technik, die man in diesem Fall anwenden kann, ist das Zurückschreiben des Inhalts des IFR, nachdem man dieses gelesen hat:

```
        STA  IFR
```

Durch diesen „Programmiertrick" wird nur dasjenige Bit zurückgesetzt, das auf „1" gesetzt war, und damit effektiv nur dieses eine Bit gelöscht, ohne daß die anderen Bits verändert werden (falls nicht mehr als ein Bit auf „1" gesetzt war).

Ein Protokoll des Quittungsbetriebs bei Eingabe

Wir wollen annehmen, daß die gesamte Befehlsfolge des Quittungsbetriebs angewendet wird: Zunächst ist das Programm dafür verantwortlich, daß ein Startimpuls („1"-Impuls) an das angeschlossene Gerät gesendet wird. Dieses wird später mit einem „Daten bereit"-Signal antworten (hier: negative Flanke). Das Programm wird dafür verantwortlich sein, festzustellen, daß das Signal empfangen wurde, und dann die Daten in den Speicher $20 zu übertragen. Hier das Programm:

```
QUITTUNG    LDA  #$00
            STA  DDRA       A IST EINGABETOR
            STA  ACR
            LDA  #$0C       BITS 2 UND 3 SETZEN
            STA  PCR        LÖSCHE STARTIMPULS
            LDA  #$0E       BITS 1, 2, 3 SETZEN
            STA  PCR        ERZEUGE STARTIMPULS
                            AUF CA2
            LDA  #$0C
            STA  PCR        LÖSCHE IHN WIEDER
WARTE       LDA  IFR        INTERRUPT?
            AND  #$02       MASKE 00000010
                            FÜR CA1-INT
            BEQ  WARTE      ABFRAGESCHLEIFE
            LDA  ORA        DATEN BEREIT
            STA  $20        ABLEGEN IN SPEICHER
```

Dieses Programm wollen wir nun im Detail studieren. Wie gewöhnlich wird Tor A als Eingabetor vereinbart, indem das DDRA mit Nullen geladen wird:

```
            LDA  #$00
            STA  DDRA       A IST EINGABETOR
            STA  ACR
```

Wir wollen annehmen, daß bei der Eingabe keine Zwischenspeicherung nötig ist (falls doch, dann ziehen Sie das vorangegangene Programm zu Rat). Nun muß das PCR derart geladen werden, daß ein „1"-Startimpuls erzeugt wird. Das Ausgangssignal von CA2 (das ist die Leitung, die wir zur Erzeugung des Startsignals verwenden werden, da CA1 nur als Eingabeleitung fungieren kann) setzen wir zuerst auf „0", dann auf „1", um einen 0/1-Übergang (positive Flanke) zu garantieren. Um den Ausgang CA2 auf „0" zu setzen, lädt man „110" in die Bits 3, 2 und 1 des PCR (Bild 2.24). Wir verwenden dazu die folgende Befehlsfolge:

```
            LDA  #$0C       BINÄR: 00001100
            STA  PCR
```

Als nächstes muß CA2 auf „1“ gesetzt werden. Hierzu laden wir „111“ in die Bits 3, 2 und 1 des PCR:

```
            LDA  #$0E       BINÄR: 00001110
            STA  PCR
```

An dieser Stelle wollen wir annehmen, daß ein kurzer Impuls als Startsignal ausreichend sei. Manche Geräte können auch Impulse längerer Dauer benötigen. In solchen Fällen müßte man an dieser Stelle ein Verzögerungsprogramm einfügen, um sicherzustellen, daß der Impuls für eine entsprechende Zeit auf „1“ bleibt. In unserem Beispiel wollen wir das CA2-Signal jedoch gleich wieder ausschalten:

```
            LDA  #$0C       BINÄR: 00001100
            STA  PCR
```

Wenn wir an diesem Punkt angelangt sind, machen wir weiter wie im vorangegangenen Programm, indem wir Bit 1 des IFR abfragen, um festzustellen, ob CA1 auf „1“ gesetzt wurde:

```
WARTE       LDA  IFR
            AND  #$02       BINÄR: 00000010
            BEQ  WARTE
```

Wie gehabt, werden die Daten dann aus dem ORA gelesen und in den Speicher $20 abgelegt:

```
            LDA  ORA
            STA  $20
```

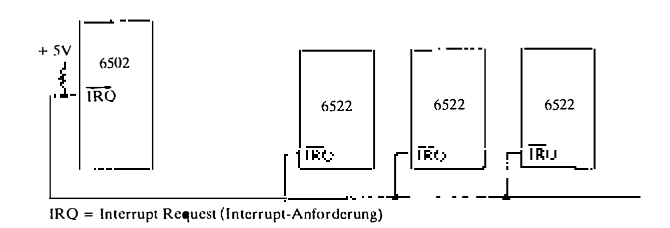


Bild 2.35: Anschluß mehrerer 6522: Interrupt-Anforderung

Verwendung mehrerer 6522

Werden mehrere 6522 gleichzeitig verwendet, werden ihre Interrupt-Ausgänge IRQ (engl.: interrupt request) an die IRQ-Leitung, wie in Bild 2.35 gezeigt, angeschlossen. Jedoch muß das Programm, wenn ein Interrupt empfangen wurde, feststellen, von welchem 6522 er kommt.

Generell wird hierfür eine Abfrageschleife benutzt. Diese wird nacheinander jedes IFR abfragen, um festzustellen, welche Einheit den Interrupt erzeugt hat. Diese Information ist mit Bit 7 des Interrupt-Flag-Registers verfügbar (Bild 2.22). Der Leser wird sich erinnern, daß das Bit 7 in Abfrageschleifen eine bevorzugte Stellung einnimmt, da das Bit 7, sobald der zu testende Registerinhalt einmal in den Akkumulator geladen ist, das Vorzeichenflag des Prozessorstatusregisters (N-Flag) festlegt. Die nächste Programminstruktion kann daher einfach das N-Flag testen und feststellen, ob es „1“ oder „0“ ist. Genau das macht die Abfrageschleife. Ein typisches Programm für eine solche Abfrageschleife lautet:

```
                LDA  IFR1
                BPL  NEXT1
INT1GEFUNDEN ···                   (IDENTIFIZIERE – EINE
.                                  VON 7 URSACHEN)
.
.
NEXT1           LDA  IFR2
                BPL  NEXT2
INT2GEFUNDEN ···                   (IDENTIFIZIERE)
.
.
.
```

Das Programm lädt den Inhalt des IFR des ersten 6522 und testet, ob dieser positiv ist. Ist er positiv, dann ist von dieser (ersten) Einheit kein Interrupt erzeugt worden und das Programm testet den nächsten 6522, etc. Ist die Einheit, von der eine Interruptanfrage vorliegt, gefunden, muß eine besondere Programmroutine bestimmen, was als nächstes zu tun ist. Das wollen wir im folgenden untersuchen.

Identifizierung eines von 7 möglichen Interrupts des 6522

Mit einem Blick auf Bild 2.22 sehen wir, daß sieben verschiedene Bedingungen möglich sind, einen internen Interrupt im IFR des 6522 zu setzen: T1, T2, CB1, CB2, SR, CA1, CA2. Wenn alle Möglichkeiten des 6522, wie es oft der Fall ist, gleichzeitig angewendet werden, sollten alle sieben Fälle getestet werden. Wir führen ein einfaches Programm vor, das einen von sieben Interrupts identifiziert:

```
EINSAUS7        ASL  A
                BMI  ZEITGEBER1
                ASL  A
                BMI  ZEITGEBER2
                ASL  A
                .
                .
                .
```

Das Programm testet nacheinander die Bits 6, 5, 4, etc., indem es den Akkumulator-Inhalt jedesmal um ein Bit nach links verschiebt. Man sollte sich im Klaren darüber sein, daß die Reihenfolge, in der der Akkumulatorinhalt geschoben wird, innerhalb jeder Einheit eine Prioritätenreihenfolgen festlegt. Indem man das Programm wie oben anwendet, gibt man dem Zeitgeber1 die höchste Priorität, dem Zeitgeber2 die zweithöchste, etc. Als Anwender möchte man möglicherweise eine andere Reihenfolge der Prioritäten festlegen. Man fragt dann die Bits in einer anderen Reihenfolge ab.

Erzeugung von Zeitverzögerungen mit einem Zeitgeber

Bevor der Leser zum ersten Mal Zeitgeberbausteine verwendet, sollte er in den Datenblättern der Hersteller deren Details studieren. Der Zeitgeber 2 ist einfacher als der Zeitgeber 1. Die beiden Zeitgeber sind nicht identisch und es ist wichtig, ihre spezifischen Charakteristika zu verstehen, bevor man sie benutzt. Da ein vollständiges Verständnis der Zeitgeber für die Belange dieses Buches nicht nötig ist, werden wir hier nur zwei typische Beispiele für die Erzeugung von Verzögerungen vorstellen, je eines für Zeitgeber 1 und 2. Andere Beispiele folgen in den Kapiteln über Anwendungen.

Erzeugung einer Einzelverzögerung mit Zeitgeber 2

Hier gleich das Programm:

```
EINZEL2     LDA  #$00
            STA  ACR        AUSWAHL BETRIEBSART
            STA  T2LL       ZWISCHENSPEICHER,
                            LOW = „0“
            LDA  #$01       VERZÖGERUNGSDAUER
            STA  T2CH       ZÄHLER,
                            HIGH = $01. START.
            LDA  #$20       MASKE
SCHLEIFE    BIT  IFR        ENDE ?
            BEQ  SCHLEIFE
            LDA  T2CL       LÖSCHT INT ZEIT-
                            GEBER 2 (T2)
```

Bits 6 und 7 des ACR müssen Null gesetzt werden, um Einzelbetrieb festzulegen (PB7 von Zeitgeber 2 nicht benutzt). Da wir hier annehmen wollen, daß keine der anderen Einheiten, wie etwa das Schieberegister, verwendet wird, laden wir einfach lauter Nullen in das ACR:

```
            LDA  #$00
            STA  ACR
```

Zeitgeber 2 enthält wie Zeitgeber 1 ein 16-Bit Zählerregister, so daß die beiden Registerhälften getrennt geladen werden müssen. Wir werden zu-

erst das niederwertige Byte (low-Byte) laden, dann das höherwertige (high-Byte):

```
          STA  T2LL
          LDA  #$01
          STA  T2CH
```

Indem man den Wert $01 in den Zähler (high-Byte) des Zeitgebers 2 (= T2CH) lädt, löscht man gleichzeitig das T2-Interrupt-Flag und startet automatisch den Zähler.

Das Bild 2.28 zeigt, daß das Bit 5 des IFR angibt, daß der Zeitgeber 2 fertiggezählt hat. Bit 5 des IFR muß daher auf den Wert „1" getestet werden. Das machen die nächsten drei Befehle:

```
          LDA  #$20       BIT5 = „1"
SCHLEIFE  BIT  IFR
          BEQ  SCHLEIFE
```

Der hexadezimale Wert 20 ist binär gleich „00100000". Mit ihm testet man, ob Bit 5 gesetzt ist. Der BIT-Befehl führt eine logische UND-Verknüpfung durch, ohne den Akkumulatorinhalt zu verändern. Solange Bit 5 „0" ist, bleibt das Programm in der Schleife und wartet auf den Interrupt von Zeitgeber 2. Sobald der Zeitgeber 2 den Interrupt erzeugt, wird dieser erkannt und das Programm verläßt die Schleife.

Schließlich muß das Programm noch explizit das Interruptflag des Zeitgebers 2 zurücksetzen, bevor es zu anderen Routinen springt. Das könnte man machen, indem man einen neuen Wert in das Zählerregister lädt. Da jedoch das Programm in allen Fällen anwendwar sein soll, werden wir keine Annahme darüber treffen, was nach der Abarbeitung dieses Programmes weiter passieren soll. Das Interrupt-Flag wird entweder durch Schreiben von T2CH oder durch Lesen von T2CL gelöscht. Da wir den Zähler nicht wieder von vorne starten wollen, werden wir nichts in das Register T2CH schreiben, sondern stattdessen T2CL lesen, einfach um den Interrupt zu löschen:

```
          LDA  T2CL
```

Erzeugung einer Einzelverzögerung mit Zeitgeber 1

Wir werden den Zeitgeber 1 hier im wesentlichen genauso verwenden, wie oben den Zeitgeber 2. Zeitgeber 1 ist jedoch im Gegensatz zu Zeitgeber 2 mit einem echten 16-Bit Zwischenspeicher ausgerüstet. Das Programm lautet dann:

```
EINZEL1   LDA  #$00
          STA  ACR        EINZELBETRIEB – PB7
                          KEIN AUSGANG
          STA  T1LL       ZWISCHENSPEICHER
                          LOW
          LDA  #$01       VERZÖGERUNGSDAUER
```

```
               STA  T1LH       ZWISCHENSPEICHER, HIGH
               STA  T1CH       LÄDT AUCH T1CL UND
                               START
               LDA  #$20
SCHLEIFE       BIT  IFR        ENDE ?
               BEQ  SCHLEIFE
               LDA  T1LL       LÖSCHT INTFLAG
```

Das Programm ist im wesentlichen analog zu dem oben vorgestellten und sollte sich daher selbst erklären. Der einzige Unterschied ist, daß zuerst der Zwischenspeicher komplett geladen wird, und dann in das höherwertige Byte des Zählers geschrieben wird. Der zweite Befehl bewirkt auch, daß der Inhalt von T1LL nach T1CL übertragen wird (siehe auch Bild 2.34, interne 6522 Register) und der Zähler startet. Der Rest des Programmes ist identisch.

Erzeugung eines Impulses

Die beiden letzten Routinen erzeugen innerhalb eines Programmes eine Zeitverzögerung. Soll hingegen ein Ausgangsimpuls erzeugt werden, muß eine passende Ausgangsleitung vereinbart werden. Für den Zeitgeber 1 wird der Pin PB7 des Tores B als Ausgangsleitung für einen Ausgangsimpuls verwendet. PB7 ist Ausgabekanal, wenn entweder DDRB7 oder ACR7 auf „1“ gesetzt ist.

Der Zeitgeber 2 sendet bei *Ausgabe* keinen direkten Impuls an einen Ausgang. Der Impuls muß daher durch zusätzliche Befehle, die eines der Bits des Tores explizit ein- und ausschalten, erzeugt werden. Jedoch kann der Zeitgeber 2 bei Impulszählbetrieb sehr einfach Impulse *zählen*. Für diesen Fall wird Pin PB6 verwendet. Dies unterstreicht nochmals die praktischen Unterschiede zwischen beiden Zeitgebern. Dem Leser wird empfohlen, bei jeder praktischen Anwendung die Datenblätter der Hersteller hinzuzuziehen, um die Bausteine optimal auszunutzen.

Ein/Ausgabe über das Schieberegister

Das Schieberegister SR ist an Pin CB2 des 6522 angeschlossen. Alle erzeugten oder empfangenen Impulse laufen über diese Leitung. Die Bits 2, 3, 4 des Hilfssteuerregisters ACR legen zusammen fest, wie das Schieberegister arbeiten soll. Bild 2.33 zeigt alle 8 Möglichkeiten.

In allen bisherigen Beispielen waren stets die Bits 2, 3, 4 des ACR auf „0“ gesetzt, so daß das Schieberegister außer Funktion war. Das serielle Ein- oder Auslesen durch das Schieberegister wird durch einen der drei möglichen Taktgeneratoren gesteuert: Zeitgeber 2, Phase 2 des Taktgenerators, oder durch einen externen Taktgenerator. Zusätzlich verfügt es über eine spezielle Betriebsart, in der der Ausgang freilaufend mit einer Frequenz getaktet wird, die durch den Zeitgeber 2 vorgegeben wird. Für eine vollständige Beschreibung des Schieberegisters sei der Leser wiederum auf die Datenblätter der Hersteller verwiesen. Wir werden in diesem Ab-

schnitt lediglich zwei typische Beispiele der seriellen Ein/Ausgabe mit dem Schieberegister vorstellen.

Eingabe mit dem Schieberegister (externer Taktgenerator)

Wir zeigen sofort das Programm:

```
SCHIEBEEIN   LDA #$00
             STA ACR        LÖSCHT SR
             LDA #$0C       EXTERNE TAKTUNG
             STA ACR        STARTET
                            SCHIEBEREGISTER
SCHLEIFE     LDA IFR        FERTIG ?
             AND #$04       TESTE BIT 2
             BEQ SCHLEIFE   WARTESCHLEIFE
             LDA SR         LIES 8 BITS ON AKKU
             STA $20        ABLEGEN IN SPEICHER
```

Das Schieberegister wird zuerst gelöscht, indem man Nullen in das ACR lädt:

```
             LDA #$00
             STA ACR
```

Dann wird die gewünschte Betriebsart definiert, indem man den Wert „011“ in die Bits 4,3,2 des ACR lädt:

```
             LDA #$0C
             STA ACR
```

Hierdurch wird die Steuerung des Schieberegisters durch einen externen Taktgenerator festgelegt (Bild 2.33).

Sobald 8 Bits getaktet sind, wird automatisch der Schiebemechanismus gesperrt und im IFR wird das SR Interruptflag gesetzt. Nachdem das Takten gestartet ist, überprüft das Programm einfach das Bit 2 des IFR auf den Wert „1“ (Bild 2.28). Hier die Abfrageschleife:

```
SCHLEIFE     LDA IFR
             AND #$04
             BEQ SCHLEIFE
```

An dieser Stelle brauchen wir den Inhalt des Schieberegisters SR nur noch wie üblich in den Speicher $20 zu übertragen:

```
             LDA SR
             STA $20
```

Ausgabe mit dem Schieberegister (gesteuert durch Phase 2)

In den Grundzügen ähnelt das Programm dem oben gezeigten. Lediglich die Steuerbits, die in das ACR geladen werden, um die entsprechende Betriebsart zu vereinbaren, sind anders. Wir nehmen an, daß wir nur ein ein-

ziges Wort bestehend aus 8 Bits ausgeben wollen, so daß am Ende keine Warteschleife nötig ist, die feststellt, ob die Ausgabeoperation beendet ist. Hier unser Programm:

```
SCHIEBEAUS   LDA  #$00
             STA  ACR      LÖSCHE SR
             LDA  #$18
             STA  ACR      Ø2 AUSGABEMODUS
             LDA  $20      LIES DATEN AUS SPEICHER
             STA  SR
```

Wie oben wird das Schieberegister auch hier zuerst gelöscht, dann wird das ACR hexadezimal mit dem Wert 18 geladen, wodurch die Bits 4,3,2 auf „110" gesetzt werden. Das legt die Steuerung des Schieberegisters durch Phase 2 des System-Taktgenerators fest:

```
LDA  #$00
STA  ACR
LDA  #$18
STA  ACR
```

Die Daten werden dann aus dem Speicher $20 geholt und in das Schieberegister übertragen. Das Ablegen der Daten im Schieberegister startet auch automatisch die Ausgabe:

```
LDA  $20
STA  SR
```

Falls wir eine ganze Reihe von 8-Bit-Worten auszugeben hätten, müßte das Programm an dieser Stelle warten, bis die Ausgabeoperation beendet wäre, bevor das nächste Datenwort in das SR übertragen würde. Man würde dies mit einer Warteschleife wie im vorherigen Beispiel machen. Sobald 8 Bits seriell ausgegeben sind setzt der 6522 automatisch das Bit 2 des IFR (Bild 2.28). Das Programm würde das Bit 2 des IFR daher einfach so lange immer wieder testen, bis es den Wert „1" annimmt. Sobald der Wert „1" gefunden wäre, könnte die serielle Ausgabe fortgesetzt werden.

Zusammenfassung des 6522

Die drei Funktionsblöcke des 6522 sind: PIO, Zeitgeber, Schieberegister. Zusätzlich können für PIO und Zeitgeber verschiedenste Steuersignale vereinbart werden. Die Funktionen der möglichen Steuersignale und Optionen wurden beschrieben. Dieser Baustein sollte als ein Satz von drei separaten Funktionseinheiten angesehen werden. Die Funktionen der Tore A und B sind grundsätzlich ähnlich, jedoch nicht symmetrisch. Die beiden Zeitgeber haben einige gemeinsame Besonderheiten, bieten jedoch unterschiedliche Möglichkeiten. Das Schieberegister schließlich ist grundsätzlich symmetrisch bei Eingabe und Ausgabe und kann zum Senden und Empfangen von Bits und Worten bei beliebiger, durch externe Taktgeneratoren vorgegebener Frequenz verwendet werden.

Übungsaufgabe 2.4: *Legen Sie in eine 2-Byte Speichertabelle mit der Adresse PUFFER zwei von GERÄT1 hintereinander ankommende Datenworte ab. GERÄT1 liefert als „Daten bereit"-Signal eine positive Flanke. Es benötigt ein Bestätigungssignal („1"-Impuls).*

Übungsaufgabe 2.5: *Genau wie 2.4, jedoch soll GERÄT1 einen „O„-Impuls als Startsignal benötigen und mit dem „Daten bereit"-Signal antworten.*

Übungsaufgabe 2.6: *Senden Sie vom Speicherbereich PUFFER Daten an GERÄT2. GERÄT2 liefert ein „belegt"-Signal, wenn es nicht bereit ist.*

Übungsaufgabe 2.7: *Genau wie 2.5, jedoch soll GERÄT2 ein STATUS-Signal benötigen, um eine bereit/belegt-Anwort geben zu können.*

Übungsaufgabe 2.8: *Schalten Sie mit einer „1" auf der Steuerleitung einen Drucker ein, warten Sie auf das „bereit"-Signal, senden Sie einen Buchstaben an den Drucker und schalten Sie diesen dann wieder aus.*

Übungsaufgabe 2.9: *Zählen Sie 10 Impulse an PB6 ab.*

Übungsaufgabe 2.10: *Erzeugen Sie an PB7 einen Impuls von 1 msec Dauer.*

Übungsaufgabe 2.11: *Geben Sie aus dem Speicher PUFFER mit Taktsteuerung durch Zeitgeber2 8 Bits aus.*

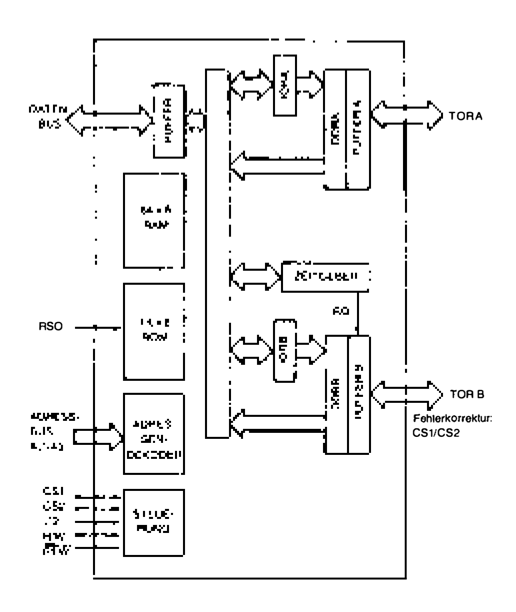


Bild 2.36: 6530 interne Architektur

Der 6530 ROM-RAM E/A Zeitgeber (RRIOT)

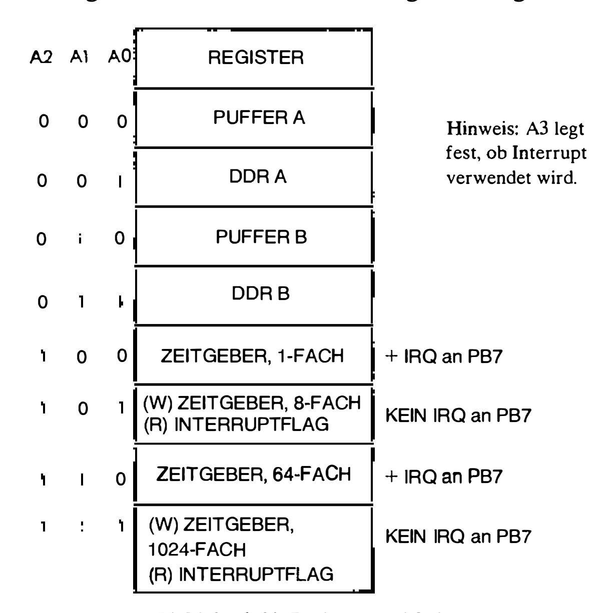
Die Abkürzung RRIOT steht für „ROM-RAM-I/O-TIMER“, das heißt so viel wie „ROM-RAM-E/A-ZEITGEBER“.

Der 6530 ist ein spezieller Kombinationsbaustein, der vier Funktionen, die gewöhnlich getrennt sind, in sich vereingt: einen PIO, einen Zeitgeber, ein ROM und ein RAM. Bild 2.36 zeigt die interne Architektur des 6530. Er ist mit den zwei normalen PIO-Toren ausgerüstet, von denen jedes ein eigenes Datenrichtungsregister hat. Jedoch stehen die Tore mit keiner der Steuerleitungen oder irgendwelcher Interruptlogik in Verbindung. Der Zeitgeber ist an Tor B angeschlossen. Der RAM-Speicher umfaßt 64 Bytes, der ROM 1 KByte. Ein ROM (read-only-memory, d. h. Nur-Lese-Speicher) kann nicht mehr verändert werden, wenn er einmal programmiert ist. Da es unwirtschaftlich ist, ROMs in kleinen Stückzahlen herzustellen, wird der 6530 nur für solche Fälle eingesetzt, wo eine große Anzahl identischer Bausteine gefertigt werden soll. Der KIM beispielsweise verwendet zwei 6530 Bausteine, in denen die internen Steuerprogramme (die Monitorprogramme) abgelegt sind.

Drei Chip-Anschlüsse haben zwei Funktionen: CS1 und CS2 sind anstelle von PB6 und PB5 Maskierungsoptionen. Ferner kann PB7 auch als Ausgang für Interruptanfragen (IRQ) Verwendung finden.

Der Intervallzeitgeber

Der Intervallzeitgeber ist mit einem 8 Bit Register ausgestattet und kann



A2	A1	A0	REGISTER	
0	0	0	PUFFER A	
0	0	1	DDR A	
0	1	0	PUFFER B	
0	1	1	DDR B	
1	0	0	ZEITGEBER, 1-FACH	+ IRQ an PB7
1	0	1	(W) ZEITGEBER, 8-FACH (R) INTERRUPTFLAG	KEIN IRQ an PB7
1	1	0	ZEITGEBER, 64-FACH	+ IRQ an PB7
1	1	1	(W) ZEITGEBER, 1024-FACH (R) INTERRUPTFLAG	KEIN IRQ an PB7

Hinweis: A3 legt fest, ob Interrupt verwendet wird.

Bild 2.37: 6530, Registerverzeichnis

in vier verschiedenen Betriebsarten verwendet werden. Je nach Wert der Bits 0 und 1 des Adreßbusses (A0 und A1) wird er in Einheiten des 1-, 8-, 64- oder 1024-fachen des Systemtaktes zählen. Dem Programmierer erscheint der Zeitgeber daher wie ein Satz von 4 Speicheradressen, wie Bild 2.37 zeigt.

Bei Verwendung des Zeitgebers kann Anschluß PB7 als Interruptausgang verwendet werden. PB7 muß dann als Eingabekanal programmiert werden. Wird er nicht als Interruptausgang verwendet, kann er ganz normal benutzt werden. Für nähere Einzelheiten über die Verwendung des PB7 als Interruptausgang wird der Leser an die Datenblätter der Hersteller verwiesen.

Der 6532 RIOT

Im wesentlichen ist der 6532 ein 6530 ohne das ROM. Dafür ist aber das RAM größer: es umfaßt 128 Bytes. Zusätzlich kann die Leitung PA7 dieses Bausteins als flankengesteuerter Eingang verwendet werden. In dieser Betriebsart setzt ein Sprung des Eingangssignals ein internes Interruptflag (Bit 6 des Interrupt-Flag-Registers).

Die interne Architektur des 6532 zeigt Bild 2.38. Bild 2.39 erläutert die Adressierung des Bausteins. Die restlichen Funktionen des 6532 sind denen des 6530 praktisch gleich.

Die Tore A und B sind nicht symmetrisch. Der Hauptunterschied zwischen ihnen ist, daß Tor B mit push-pull-Puffern ausgerüstet ist, die bei 1,5 V bis zu 3 mA ziehen können. Das erlaubt den direkten Anschluß von LEDs oder Darlington-Transistoren an dieses Tor. Ferner liest Tor A die Eingangssignale direkt ein, während Tor B sie stattdessen aus dem Ausgaberegister liest.

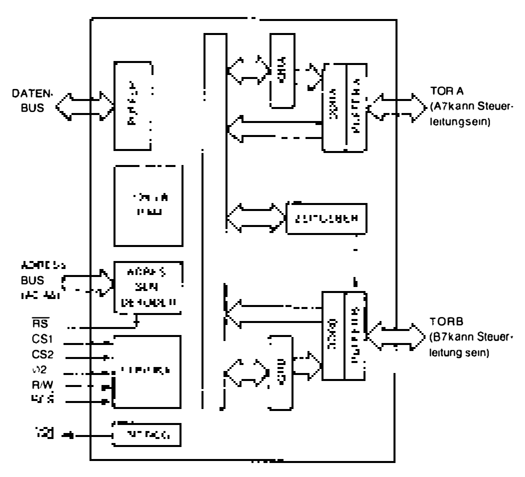


Bild 2.38: 6532, interne Architektur

RS	A4	A3	A2	A1	A0	R/W	REGISTERWAHL
0	-	.	-	-	-	.	RAM
1	.	.	0	0	0	.	ORA
1	.	-	0	0	1	.	DDRA
1	.	.	0	1	0		ORB
1	.		0	1	1	.	DDRB
1	1	*	1	0	0	0	LADE ZEITGEBER (1-FACH)
1	1	*	1	0	1	0	LADE ZEITGEBER (8-FACH)
1	1	*	1	1	0	0	LADE ZEITGEBER (64-FACH)
1	1	*	1	1	1	0	LADE ZEITGEBER (1024-FACH)
1		-	1		0	1	LIES ZEITGEBERREGISTER
1	.	.	1	.	1	1	LIES INTERRUPTFLAG
1	0		1	**	***	0	SETZE FLANKENSTEUERUNG

* Sperren(0)/Zulassen(1) des INT vom Zeitgeber nach IRQ

** Sperren(0)/Zulassen(1) des INT von PA7 nach IRQ

*** negative(0)/positive(1) Flankensteuerung

Bild 2.39: Adressierung des 6532

	6520	6522	6530	6532
LEITUNGEN TOR A	8	8	8	8
LEITUNGEN TOR B	8	8	5 bis 8	8
STEUERLEITUNGEN A	2	2	0	0
STEUERLEITUNGEN B	2	2	0	0
DDRA	1	1	JA	JA
DDRB	1	1	JA	JA
ZEITGEBER 1	.	JA	JA	JA
ZEITGEBER 2	.	JA	.	.
ROM		.	1K x 8	.
RAM	.	-	64 x 8	128 x 8
SONSTIGES	.	zus. Steuerregister	4 Zeitg.frequ.	4 Zeitg.frequ.
INTERRUPTS	2	1	als Option	1

Bild 2.40: Vergleichsübersicht über die vier PIO's

Zusammenfassung

Die meisten Anwendungen werden mindestens zwei oder mehr Tore an einer oder mehreren PIOs sowie den Einsatz eines programmierbaren Zeitgebers verlangen. Komplexere Anwendungen bedingen die Verwendung von Steuersignalen und möglicherweise den Einsatz automatischer parallel/seriell Wandler. Alle betrachteten Bausteine – der 6520, der 6522, der 6530 und der 6532 – verfügen über zwei PIO-Tore. Mit Ausnahme des 6520 hat jeder Baustein mindestens einen programmierbaren Zeitgeber. Eine Vergleichsübersicht über die vier Ein/Ausgabebausteine zeigt Bild 2.40.

Alle in diesem Buch beschriebenen Anwendungen werden einen oder mehrere der in diesem Kapitel beschriebenen PIOs einsetzen.

Kapitel 3

6502 Systeme

Einleitung

Die in diesem Buch vorgestellten Anwendungen werden sich auf ein „Standard"-6502-System beziehen. Daher wollen wir die Organisation eines solchen „Standard-Systems" als erstes vorstellen. Anschließend beschreiben wir einige existierende 6502-Mikrocomputer und zeigen, daß sie weitgehend mit dem eingangs beschriebenen Standardmodell in Einklang stehen.

Um realistische Anwendungen darzustellen, muß man notwendigerweise eine genaue Hardwarekonfiguration definieren, auf die sich alle Anwendungen beziehen. Die überwiegende Zahl der Beispiele läßt sich direkt auf den SYM und praktisch genauso auf den KIM anwenden. In einem Abschnitt des nächsten Kapitels werden in erster Linie KIM-Programme vorgestellt. SYBEX empfiehlt keinen speziellen Mikrocomputer oder Hersteller. Einzig aus pädagogischen Gründen ist es angebracht, die Anwendungen so zu gestalten, daß sie ablaufen können auf existierenden Geräten, ohne daß extra ein fiktiver Mikrocomputer vereinbart werden müßte. Die meisten für den SYM geschriebenen Programme sind KIM-kompatibel und können praktisch genauso auf anderen Systemen verwendet werden. Der Leser sollte selbst entscheiden, welcher Mikrocomputer für seine speziellen Bedürfnisse am besten geeignet ist.

In diesem Kapitel stellen wir die Architektur von KIM, SYM, AIM65, PET/CBM, VC20 und APPLE II vor. SYM wird detaillierter erklärt, damit auch derjenige Leser, der keinen SYM besitzt, die in den Anwendungen der nächsten Kapitel beschriebenen Schaltungen verstehen kann. Es soll jedoch nochmals deutlich betont werden, daß jeder andere 6502-Mi-

krocomputer verwendet werden kann und daß die notwendigen Änderungen minimal sind.

Ein „Standard“-6502-System

Jedes Standardsystem umfaßt zumindest die zentrale Prozessoreinheit (CPU, central processor unit) mit zugehörigem Taktgeber (clock circuit), sowie ROM, RAM und einen oder mehrere PIO's. Die Organisation eines solchen Standardsystems mit einem 6502 zeigt Bild 3.1.

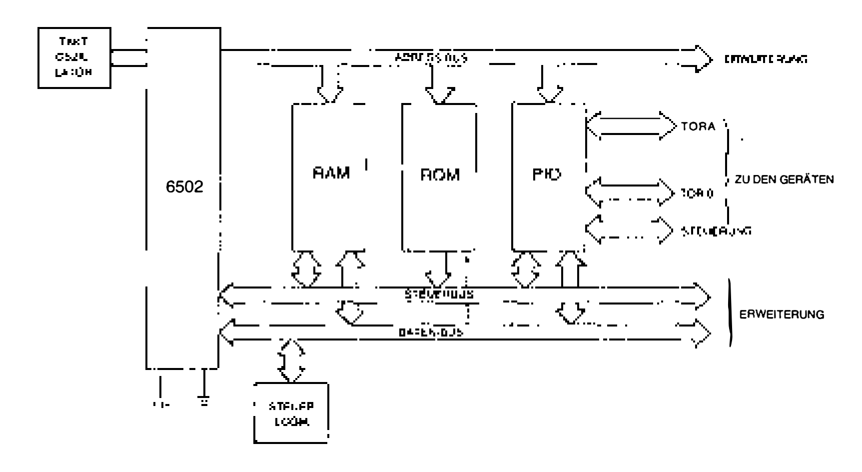


Bild 3.1: Organisation eines „Standard“-6502-Systems

Der größte Teil der Taktgeberschaltung ist schon auf dem 6502-Chip enthalten, so daß nur noch der externe Schwingquarz und die Oszillatorschaltung benötigt werden. Der linke Teil von Bild 3.1 zeigt den 6502 und seinen Taktoszillator. Vom 6502 werden – wie von jedem Standardmikroprozessor – drei Busse erzeugt: der Adreßbus (16 Leitungen), der Datenbus (8 Leitungen, bidirektional) und schließlich der Steuerbus.

In unserem Standardsystem werden der RAM-Speicher (Schreib-Lese-Speicher), der ROM-Speicher (Nur-Lese-Speicher) und der PIO (parallele E/A) als separate, an die drei Busse angeschlossene Einheiten dargestellt. Für gewöhnlich wird das ROM ein *Monitor*-Programm enthalten, das für den Betrieb des Mikrocomputers nötig ist, oder auch (besonders bei industriellen Anwendungen) Anwenderprogramme. Der PIO hat zwei Tore zu je 8 Bit für die Kommunikation mit der Peripherie, außerdem eventuelle zusätzliche Steuerleitungen. Bei jeder praktischen Anwendung werden mindestens zwei PIO's nötig sein, um eine ausreichende Zahl von E/A-Leitungen bereitzustellen. Normalerweise werden einige

zusätzliche Logikbausteine zur Adreß-Dekodierung oder für andere Funktionen benötigt.

Da in der 6502-Familie mehrere Kombinationsbausteine zur Verfügung stehen, können ROM, RAM und PIO auch in einem oder mehreren Bausteinen zusammengefaßt sein. Jedes System, das den 6502 verwendet, wird aber im allgemeinen alle logischen Elemente aus Bild 3.1 enthalten.

Wir wollen nun einige existierende Mikrocomputer untersuchen und uns überlegen, inwieweit sie mit unserem Standardsystem übereinstimmen.

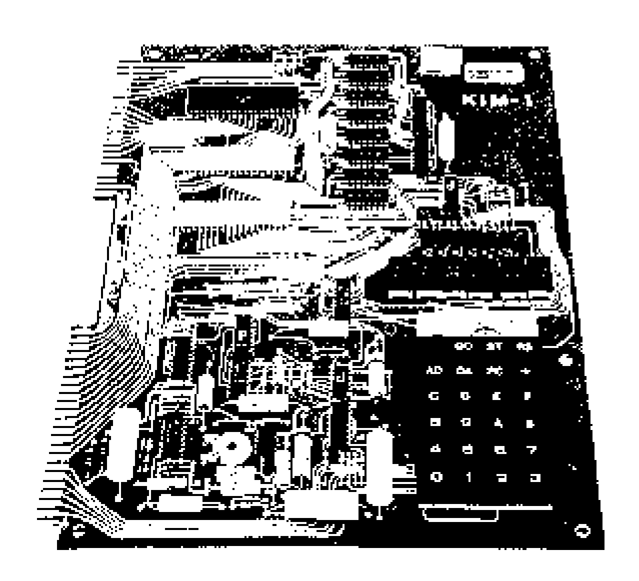


Bild 3.2: Der KIM-1

Der KIM-1

Der KIM-1 war ein früher Mikrocomputer, der von MOS Technology zur Unterstützung ihres 6502 Mikroprozessors eingeführt wurde. Er enthält nur eine Mindestzahl von Bauteilen, ist mit einer hexadezimalen Tastatur und mit 6 LED's ausgerüstet, und kann als preiswerter, vollständiger, alleine einsetzbarer Mikrocomputer verwendet werden. Er ist in Bild 3.2 gezeigt. Seine interne Organisation sehen Sie in Bild 3.3.

Der KIM-1 enthält ein separates 1K mal 8 RAM (für den Benutzer) und zwei 6530 Kombinationsbausteine. Aus dem vorangehenden Kapitel wissen wir, daß der 6530 ein Kombinationsbaustein ist, der eine PIO, einen programmierbaren Zeitgeber, ein ROM und ein RAM enthält. Da die ROMs, die die beiden 6530 zur Verfügung stellen, ausreichen, um den Sy-

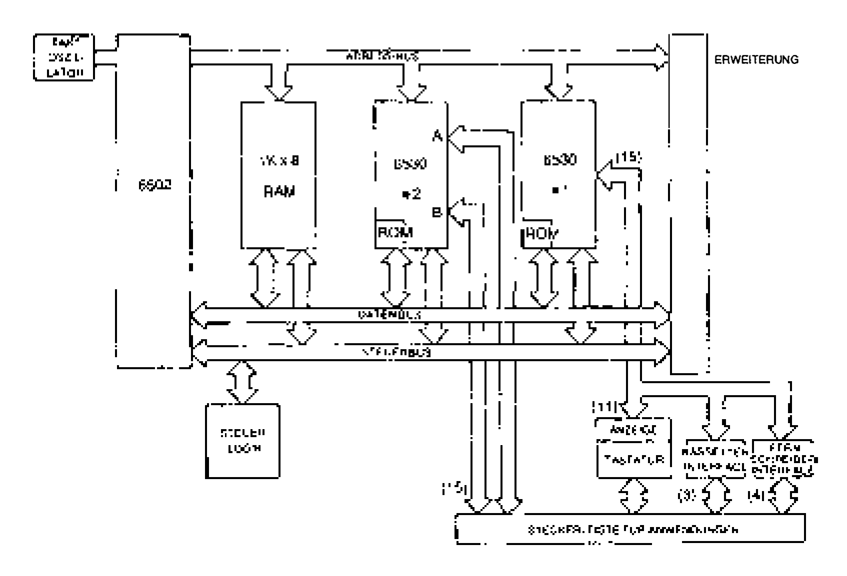


Bild 3.3: Interne Organisation des KIM-1

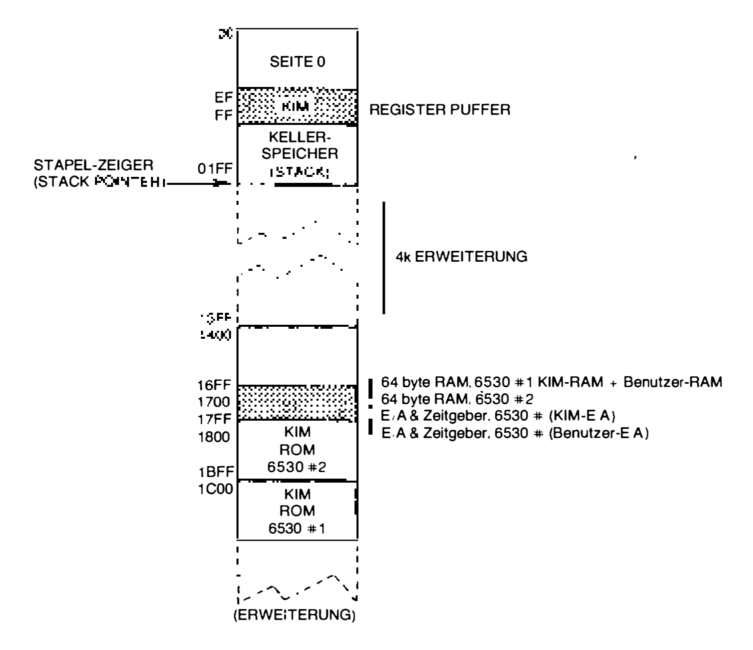


Bild 3.4: KIM-1 Speicheraufteilung

stemmonitor aufzunehmen, werden auf dieser Mikrocomputerplatine keine externen ROMs benötigt. Jeder 6530 enthält außerdem noch 64 Byte RAM-Bereiche, die teilweise vom Systemmonitor belegt werden.

Zusätzlich sind auf der Platine eine Tastatur, 6 LED's, ein Kassetten-Interface und ein Fernschreiber-Interface vorhanden. Über zwei Kontaktleisten kann sie extern erweitert werden. Dies sind der Erweiterungsstekker und der Anwenderstecker, wie Bild 3.3 zeigt. Die Speicheraufteilung des Systems zeigt Bild 3.4. Die Bedeutung der verschiedenen Anschlüsse der beiden Steckerleisten zeigen Bild 3.5 und 3.6.

Der Leser sollte sich überzeugen, daß die Organisation dieses Mikrocomputers mit der Beschreibung unseres Standardsystems aus Bild 3.1 übereinstimmt. Die Einzelheiten der Anschlußbelegung sind für diejenigen Leser von Nutzen, die die später vorgestellten Anwendungsbeispiele auf diesem Mikrocomputer anwenden wollen.

22	KB Spalte D	Z	KB Reihe 1
21	KB Spalte A	Y	KB Spalte C
20	KB Spalte E	X	KB Reihe 2
19	KB Spalte B	W	KB Spalte G
18	KB Spalte F	V	KB Reihe 3
17	KB Reihe 0	U	TTY PTR
16	PB5	T	TTY KYBD
15	PB7	S	TTY PTR RTRN (+)
14	PA0	R	TTY KYBD RTRN (+)
13	PB4	P	AUDIO OUT HI
12	PB3	N	+12V
11	PB2	M	AUDIO OUT LO
10	PB1	L	AUDIO IN
9	PB0	K	DECODE ENAB
8	PA7	J	K7
7	PA6	H	K5
6	PA5	F	K4
5	PA4	E	K3
4	PA1	D	K2
3	PA2	C	K1
2	PA3	B	K0
1	V_{SS} (GND)	A	V_{CC} (+5V)

Bild 3.5: KIM Anwender-Steckerleiste

22	V_{SS} (GND)	Z	RAM/R/W
21	V_{CC} (+5)	Y	$\overline{\phi 2}$
20		X	PLL TEST
19		W	$\overline{R/W}$
18		V	R/W
17	SST OUT	U	$\phi 2$
16	K6	T	AB15
15	DB0	S	AB14
14	DB1	R	AB13
13	DB2	P	AB12
12	DB3	N	AB11
11	DB4	M	AB10
10	DB5	L	AB9
9	DB6	K	AB8
8	DB7	J	AB7
7	RST	H	AB6
6	NMI	F	AB5
5	RO	E	AB4
4	IRQ	D	AB3
3	$\phi 1$	C	AB2
2	RDY	B	AB1
1	SYNC	A	AB0

Bild 3.6: KIM Erweiterungsstecker

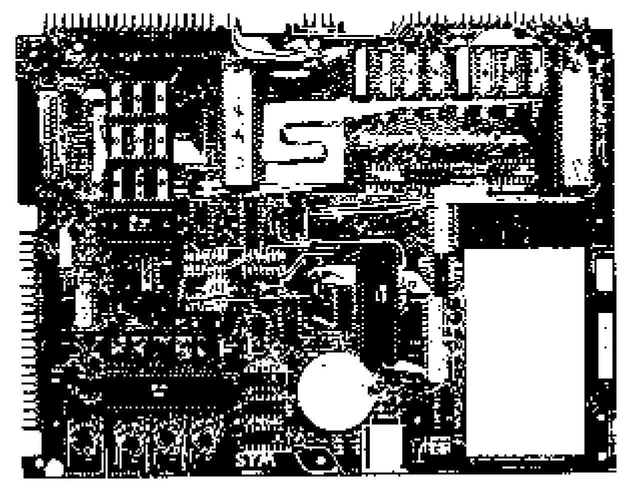


Bild 3.7: Der SYM

Der SYM-1

Der Mikrocomputer SYM-1 wurde von Synertek Systems als erweiterte Version des vorhergehenden Mikrocomputers eingeführt. Bild 3.7 zeigt ein Foto des SYM. Seine interne Organisation zeigt Bild 3.8.

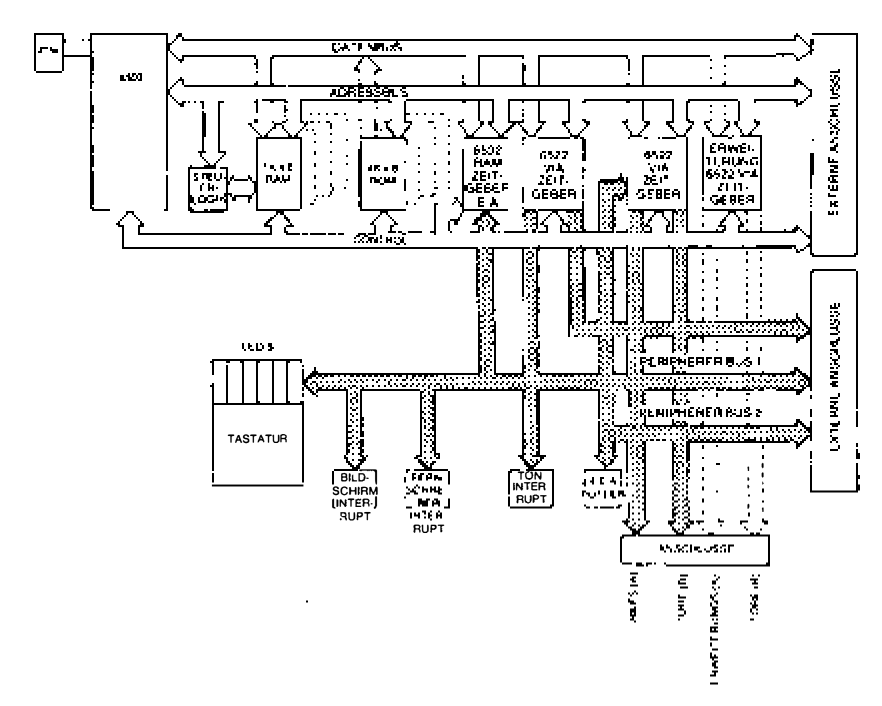


Bild 3.8: Interne Organisation des SYM

- Die entscheidenden Unterschiede zum vorhergehenden Mikrocomputer sind:
- Er ist mit einem separaten 4K X 8 ROM ausgerüstet. Ein größerer ROM-Bereich erlaubt die Unterbringung eines komplexeren Monitorprogramms.
- Er ist mit komlexeren Ein/Ausgabe-Bausteinen ausgestattet. Anstelle von nur zwei hat er drei und bietet dadurch eine größere Zahl E/A-Tore etc. Durch die zusätzlichen Tore hat er eine Steckerleiste mehr für den Benutzer als die vorhergehende Platine.
- Es sind zusätzliche E/A-Möglichkeiten verfügbar, wie etwa vier E/A-Puffer und Teile eines Kassetteninterface.

Es gibt noch viele andere Unterschiede, die jedoch für die Belange dieses Buches unerheblich sind.

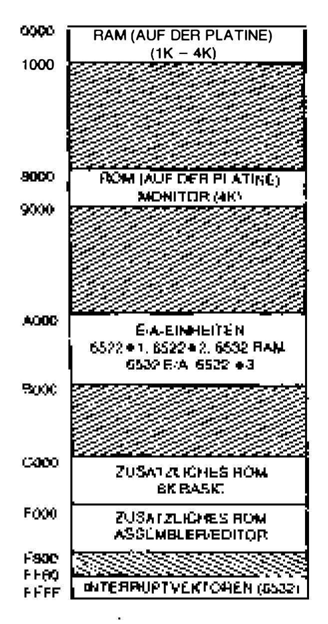


Bild 3.9: System-Speicheraufteilung

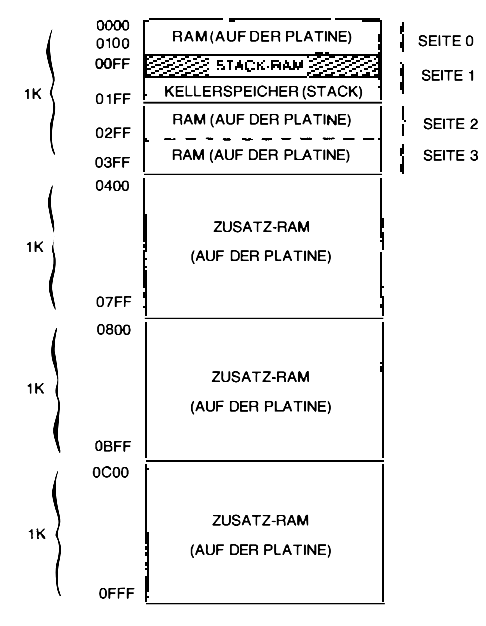


Bild 3.10: RAM-Speicheraufteilung

Die Speicheraufteilung des Systems zeigt Bild 3.9, eine detaillierter herausgezeichnete RAM-Speicheraufteilung finden Sie in Bild 3.10. Die Belegung der drei Steckerleisten zeigen die Bilder 3.11, 3.12 und 3.13.

	Pin	Signal	Pin	Signal
	1	SYNC	A	AB0
	2	RDY	B	AB1
	3	$\overline{Ø1}$	C	AB2
	4	$\overline{IRQ}$	D	AB3
	5	RO	E	AB4
	6	$\overline{NMI}$	F	AB5
	7	$\overline{RES}$	H	AB6
	8	DB7	J	AB7
	9	DB6	K	AB8
	10	DB5	L	AB9
	11	DB4	M	AB10
	12	DB3	N	AB11
	13	DB2	P	AB12
	14	DB1	R	AB13
	15	DB0	S	AB14
	16	$\overline{18}$	T	AB15
	17	DBOUT (1)	U	$\overline{Ø2}$
	18	$\overline{POR}$	V	R/W
Nicht belegt	19	Unused	W	$\overline{R/W}$
Nicht belegt	20	Unused	X	AUD TEST
	21	+5V	Y	$\overline{Ø2}$
	22	GND	Z	RAM - R/W

Bild 3.11: Erweiterungs-Steckerleiste (E)

Pin	Signal	Pin	Signal
1	GND	A	+5V
2	APA3	B	$\overline{00}$
3	APA2	C	$\overline{04}$
4	APA1	D	$\overline{08}$
5	APA4	E	$\overline{0C}$
6	APA5	F	$\overline{10}$
7	APA6	H	$\overline{14}$
8	APA7	J	$\overline{1C}$
9	APB0	K	$\overline{18}$
10	APB1	L	Audio In
11	APB2	M	Audio Out (LO)
12	APB3	N	RCN-1 (1)

Bild 3.12: Anwender-Steckerleiste (A)

13	APB4	P	Audio Out (HI)
14	APA0	R	TTY KB RTN (+)
15	APB7	S	TTY PTR (+)
16	APB5	T	TTY KB RTN (−)
17	TASTATUR ZEILE 0	U	TTY PTR (−)
18	TASTATUR SPALTE F	V	TASTATUR ZEILE 3
19	TASTATUR SPALTE B	W	TASTATUR SPALTE G
10	TASTATUR SPALTE E	X	TASTATUR ZEILE 2
21	TASTATUR SPALTE A	Y	TASTATUR SPALTE C
22	TASTATUR SPALTE D	Z	TASTATUR ZEILE 1

(1): Jumper Option

Bild 3.12: Anwender-Steckerleiste (A)

1	GND	A	+5V
2	$-V_N$	B	$+V_P$
3	2 PA 1	C	2 PA 2
4	2 CA 2	D	2 PA 0
5	2 CB 2	E	2 CA 1
6	2 PB 7	F	2 CB 2
7	2 PB 5	H	2 PB 6
8	2 PB 3	J	2 PB 4
9	2 PB 1	K	2 PB 2
10	2 PA 7	L	2 PB 0
11	2 PA 5	M	2 PA 6
12	2 PA 3	N	2 PA 4
13	RES	P	3 CA 1
14	3 CB 1	R	SCOPE
15	3 PB 2	S	3 PB 3
16	3 PB 0	T	3 PB 1
17	3 PA 6	U	3 PA 7
18	3 PA 3	V	3 PA 0
19	3 PA 4	W	3 PA 1
20	3 PA 5	X	3 PA 2
21	3 PB 5 (B)	Y	3 PB 4 (B)
22	3 PB 7 (B)	Z	3 PB 6 (B)

(P): Gepuffert

Bild 3.13: Hilfs-Anwendersteckerleiste (AA)

Die Speicheraufteilung für die 6522-Bausteine finden Sie in Bild 3.14, das für die 6532-Bausteine in Bild 3.15.

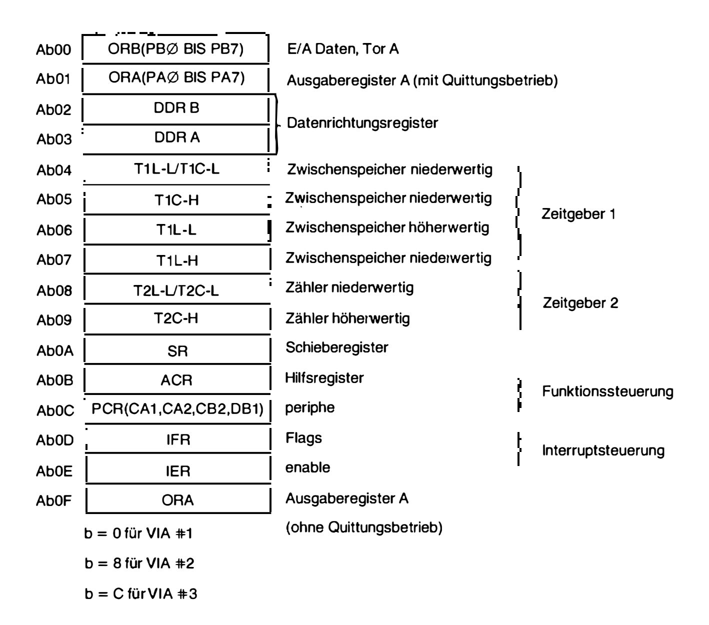


Bild 3.14: Speicheraufteilung für die 6522-Bausteine

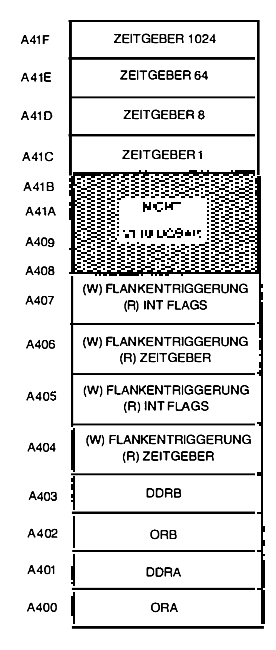


Bild 3.15: Speicheraufteilung für 6532

Da in manchen Anwenderprogrammen einige spezifische Eigenarten des SYM benötigt werden, wollen wir im folgenden zwei dieser Feinheiten vorstellen.

Bild 3.16 zeigt die vier gepufferten Ausgänge PB4 bis PB7, die am 6522 $3 verfügbar sind. In Bild 3.17 sehen Sie, wie die LED's und die Tastatur angeschlossen sind.

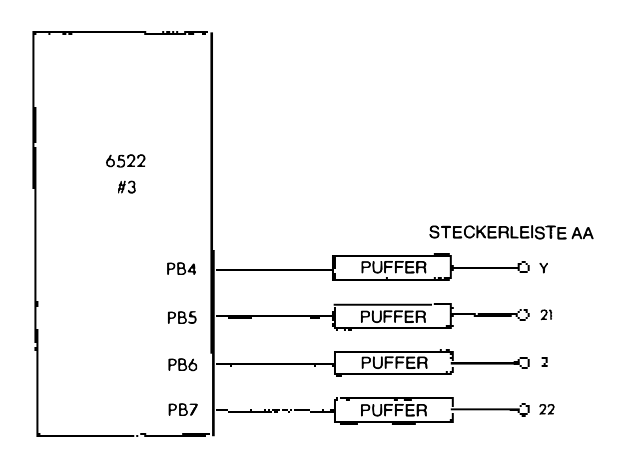


Bild 3.16: Die vier gepufferten Ausgänge

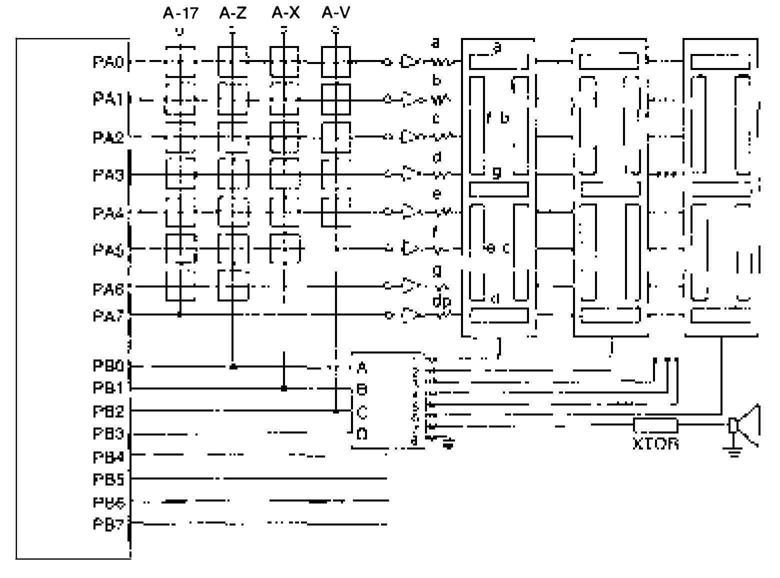


Bild 3.17: Anschluß von Tastatur und LED's

Der AIM65

Den AIM65 zeigt Bild 3.18. Dieses von Rockwell International entwikkelte Gerät besteht aus zwei Platinen. Die eine enthält den Mikrocomputer und ist mit einem zwanzigspaltigen Punktmatrixdrucker und einer zwanzig Buchstaben umfassenden alphanumerischen Anzeige ausgestattet. Die zweite Platine ist eine komplette ASCII-Tastatur. Sie wird direkt an die andere Platine angeschlossen. Der Drucker druckt maximal



Bild 3.18: Der AIM65 mit Mini-Drucker und vollständiger Tastatur

120 Zeilen pro Minute. Mit einer fünf-mal-sieben Punktmatrix stellt er den kompletten Satz von 64 ASCII-Zeichen dar. In seiner Minimalversion ist der AIM65 mit einem komfortablen 8K-Monitor, einem RAM-Bereich von 1K, zwei 6522, einem 6532 und den üblichen Interfaces (Fernschreiber, zwei Kassetteninterfaces und natürlich das Tastaturinterface) ausgerüstet. Auf der Platine können problemlos etliche Zusatz-Chips eingefügt werden. Ferner ist die Anschlußbelebung der Benutzer-Steckerleiste dieselbe wie bei den vorangehenden Mikrocomputern. Ein Anwender, der für diesen speziellen Mikrocomputer Programme entwikkelt, muß daher die hier vorgestellten Programme nur so ändern, daß die Speicherbelegung der PIO's des AIM65 wieder paßt.

PET/CBM

Seit Ende der Siebziger Jahre wurde von COMMODORE sehr erfolgreich eine Serie von Mikrocomputern auf den europäischen Markt gebracht. Als erstes Gerät wurde 1977 der PET 2001 eingeführt, der schon bald durch die CBM's der Serien 3000, 4000 und zuletzt 8000 abgelöst wurde. Alle diese Geräte sind BASIC-programmierbare, vollständige Mikrocomputer, die über eine ASCII-Tastatur (8000 DIN), sowie einen eingebauten TV-Monitor verfügen. Das Bild 3.19 zeigt als Beispiel einen CBM der 8000'er Serie. Die Rückkseite des Gerätes ist schematisch in Bild 3.20 zu sehen.

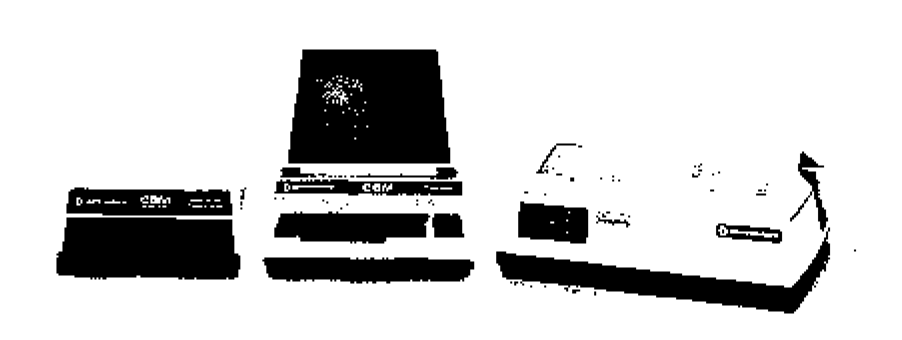


Bild 3.19: CBM 8032 mit Floppy-Disk (links) und Drucker (rechts)

Alle Geräte umfassen zwischen 8 und 32 KByte RAM's, von denen etwa 1 KByte für interne Aufgaben reserviert sind, sowie 18 KByte ROM's, die das Betriebssystem und den BASIC-Interpreter fassen, ferner einige Ein/Ausgabe-Bausteine.

Für Ein/Ausgabe stehen dem Benutzer ein Kassetten-Interface, ein IEEE-488-Bus und der sogenannte *USER-port* zur Verfügung. Für die Belange dieses Buches ist nur der USER-port von Bedeutung. Es handelt sich um eine Steckerleiste mit 12 Doppelkontakten. Hiervon sind die 12 unteren Kontakte mit dem Tor A sowie den Steuerleitungen CA1 und CB2 eines VIA 6522 verbunden. Den Blick auf den USER-port zeigt Bild 3.21, die Belegung der Kontakte Bild 3.22.

Alle E/A-Leitungen des USER-port sind für den Benutzer frei verfügbar. Die restlichen E/A-Leitungen dieses VIA, die nicht über den USER-port zugänglich sind, sind schon intern belegt, und dürfen nicht benutzt werden. Die zu diesem VIA gehörende Speicheraufteilung zeigt Bild 3.23.

Bei Verwendung von Routinen des Betriebssystems gestalten sich die Dateneingabe über die Tastatur sowie die Datenausgabe über den Bildschirm recht einfach. Die wichtigsten dieser Routinen sind:

BSOUT	$FFD2	AUSGABE EINES ASCII-ZEICHENS IM AKKUMULATOR ÜBER DEN BILDSCHIRM
BASIN	$FFCF	INPUT-ROUTINE. EINGABEENDE BEI ‚RETURN'.
GETIN	$FFE4	GET-ROUTINE. EINGABE EINES EINZELNEN ASCII-ZEICHENS IN DEN AKKUMULATOR. KEIN ZEICHEN = $00.

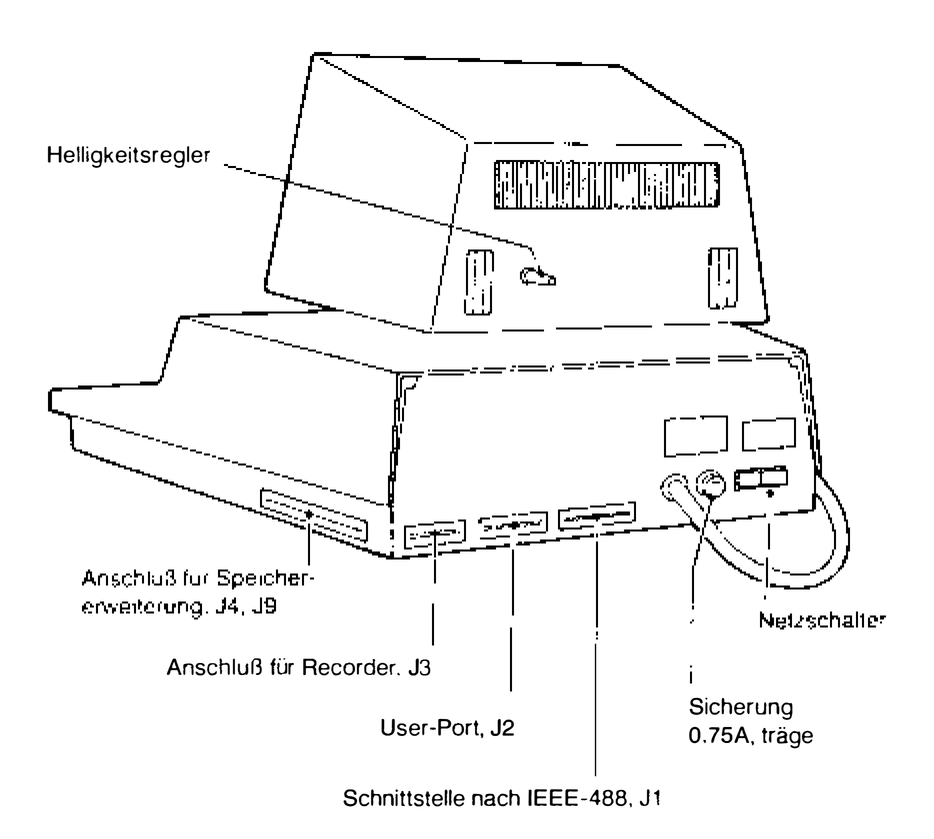


Bild 3.20: Rückseite des CBM 4032

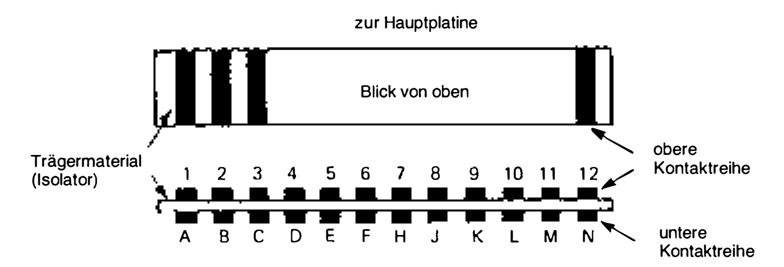


Bild 3.21: Blick auf den USER-port und Kontaktbenennung

Kontakt	Signal	Beschreibung
OBEN:		
1.	GND	Masse (Digital)
Codierschlitz		
2.	TV-Video	Video-Ausgang für externen Bildschirm.
3.	IEEE-SRQ	Für Testroutine
4.	IEEE-EOI	Für Testroutine
5.	Diagnose?	Testroutine wird gerufen, wenn 'low' beim Einschalten
6.	§ 1 READ	Überprüfung der Lesefunktion von Rekorder 1
7.	§1,§2 WRITE	Überprüfung der Schreibfunktion beider Rekorderanschlüsse
8.	§ 2 READ	Überprüfung der Lesefunktion von Rekorder 2
9.	TV-VERT	Vertikalsynchronsignal (60 Hz)
10.	TV-HOR	Horizontalsignal
Codierschlitz		
11.	GND	Masse (Digital)
12.	GND	Masse (Digital)
UNTEN:		
A.	GND	Masse (Digital)
Codierschlitz		
B.	CA1	Flankengetriggerter Anschluß des VIA 6522.
C.	PA0	
D.	PA1	
E.	PA2	
F.	PA3	PA0 bis PA7 sind einzeln als Eingangs- oder
H.	PA4	Ausgangsleitungen programmierbar. Sie sind
J.	PA5	direkt mit dem VIA 6522 verbunden.
K.	PA6	
L.	PA7	
Codierschlitz		
M.	CB2	Anschluß CB2 des VIA 6522
N.	GND	Masse (Digital)

Bild 3.22: Kontaktbelegung des USER-port

Dez.	Hex.	Adressierte Speicherstelle/Tätigkeit
59456	E840	Output Register für I/O-Port B (ORB)
59457	E841	Output Register für I/O-Port A (ORA) mit Handshake
59458	E842	I/O-Port B Datenrichtungsregister (DDRB)
59459	E843	I/O-Port A Datenrichtungsregister (DDRA)
59460	E844	Lies LSB des Zählers im Timer 1 und schreibe es als LSB ins Latch vom Timer 1.
59461	E845	Lies MSB des Zählers im Timer 1, schreibe es als MSB ins Latch vom Timer 1 und beginne zu zählen
59462	E846	Hole LSB vom Latch des Timers 1.
59463	E847	Hole MSB vom Latch des Timers 1.
59464	E848	Lies LSB des Timers 2 und setze IFR zurück. Schreibe es als LSB ins Latch vom Timer 2 ohne Reset
59465	E849	Lies MSB des Timers 2. Reset IFR beim Schreiben ins MSB des Latch.
59466	E84A	Serielles I/O-Schieberegister (SR)
59467	E84B	Hilfskontrollregister (ACR)
59468	E84C	Peripherie-Kontrollregister (PCR)
59469	E84D	Interrupt Flag-Register (IFR)
59470	E84E	Interrupt Enable-Register (IER)
59471	E84F	Output Register für I/O-Port A ohne Handshake

Bild 3.23: Speicheraufteilung des VIA 6522 (PET/CBM)

VC-20

Im Jahr 1981 erweiterte COMMODORE sein Computer-Programm durch den VC-20, den sogenannten ‚Volkscomputer'. Der VC-20 bietet dem Anwender im wesentlichen dieselben Möglichkeiten wie PET und

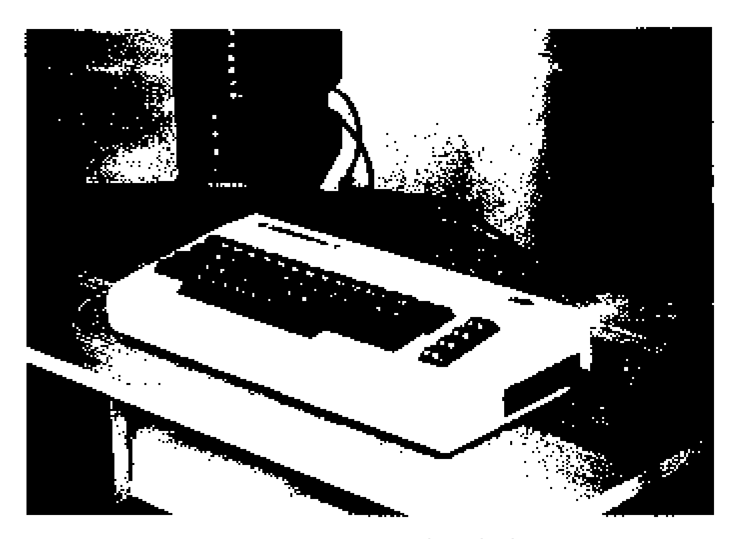


Bild 3.24: Der VC-20 von COMMODORE

CBM, jedoch fehlt der eingebaute Bildschirm. Der VC-20 kann aber an jedes Fernsehgerät oder an jeden Bildschirm angeschlossen werden. Bild 3.24 zeigt den VC-20, Bild 3.25 die Rückansicht.

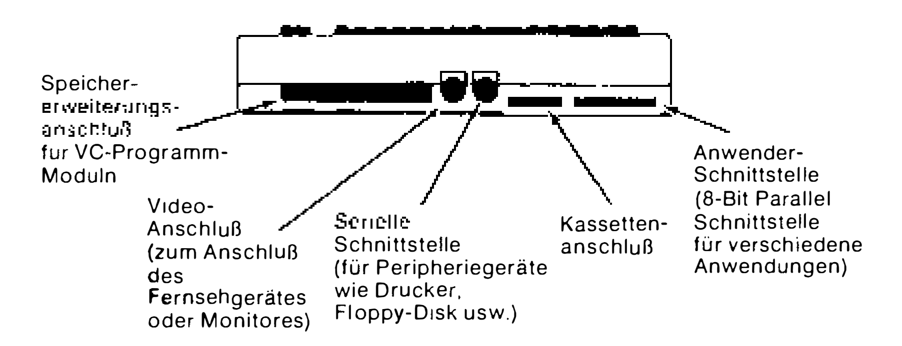
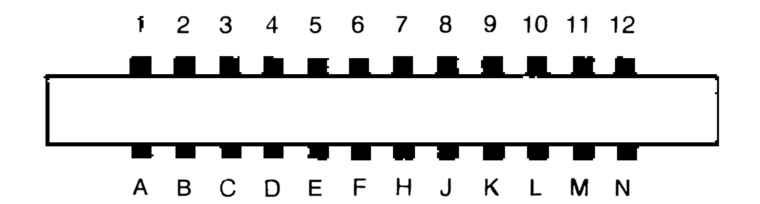


Bild 3.25: Rückansicht des VC-20, schematisch



PIN	TYPE	BEMERKUNG	PIN ÷	TYPE	BEMERKUNG
1	GND		A	GND	
2	+ 5V	100mA MAX.	B	CB1	
3	RESET		C	PB0	
4	JOY0		D	PB1	
5	JOY1		E	PB2	
6	JOY2		F	PB3	
7	LIGHT PEN		H	PB4	
8	CASSETTE SWITCH		J	PB5	
9	SERIAL ATN IN		K	PB6	
10	+ 9V	100mA MAX.	L	PB7	
11	GND		M	CB2	
12	GND		N	GND	

Bild 3.26: USER-port des VC-20, Kontaktbelegung

In seiner Grundversion verfügt der VC-20 über 3,5 KByte RAM, 20 KByte ROM mit Betriebssystem und BASIC-Interpreter, zwei VIA 6522, sowie Ein/Ausgabe-Bausteine für einen TV-Ausgang und Spiele-Steuerungen. Wie bei den schon beschriebenen PET und CBM hat der Benutzer über einen sogenannten USER-port Zugang zu einem 8-Bit-Tor (hier jedoch Tor B), einem der beiden VIA's, sowie die beiden Steuerleitungen CB1 und CB2. Die Kontaktbelegung zeigt Bild 3.26. Der VIA des USER-Ports hat die Adresse $9110.

APPLE II

Bild 3.27 zeigt den APPLE II mit Monitor und zwei Diskettenlaufwerken, Bild 3.28 den Blick in das Innere des Gerätes. In das Gehäuse ist eine vollständige ASCII-Tastatur integriert. Die Datenausgabe erfolgt im Dialogbetrieb meist über einen extern angeschlossenen Fernsehbildschirm oder Monitor (Bild 3.27).

Der APPLE II verfügt über eine Speicherkapazität von 48 KByte RAM sowie maximal 12 KByte ROM für den Systemmonitor, den BASIC-Interpreter und Hilfsprogramme. Als einziges der vorgestellten Geräte besitzt er in der Grundversion keine Ein/Ausgabe-Bausteine. Im hinteren Teil der Hauptplatine befinden sich aber 8 Kontaktleisten („slots"), in die Ein/Ausgabe-Platinen gesteckt werden können (Bild 3.28). Die Kontaktbelegung dieser slots zeigt Bild 3.29. Jedem slot sind für Ein/Ausgabeoperationen 16 Adressen zugewiesen, ferner je 256 Byte RAM-Speicherplatz. Eine Übersicht über die E/A-Adressen zeigt Bild 3.30. Weitergehende Einzelheiten können dem *APPLE II reference manual* entnommen werden.

Es läßt sich leicht eine Schnittstelle mit dem VIA 6522 für den APPLE bauen, die für jeden slot verwendbar ist. Die Registeradressen entnimmt man Bild 3.30. Den Verdrahtungsplan zeigt Bild 3.31. Die Bezeichnungen rechts im Bild beziehen sich auf die Kontaktbelegung der slots nach Bild 3.29. Eine Schwierigkeit stellt hierbei der Anschluß Φ 2 dar, der über einen Doppelinverter (2 NAND-Gatter) als Treiber an den VIA angeschlossen werden muß, da er über die slots nicht verfügbar ist. Man kann diesen Ausgang Φ 2 direkt von der CPU über eine separate Leitung der Interfaceplatine zuführen. Eleganter, aber auch aufwendiger, führt man das Taktsignal Φ 2 über einen der beiden nicht belegten Kontakte eines slots (19 oder 35) von der Unterseite der Hauptplatine der VIA-Platine zu. Die entsprechenden Leiterbahnen auf der Hauptplatine, die einige der slot-Kontakte miteinander verbinden, trennt man bei dem betreffenden slot sicherheitshalber auf. Dem Anfänger ist von diesem Eingriff jedoch abzuraten. Bild 3.32 zeigt einen Probeaufbau einer derartigen 6522-Schnittstelle.

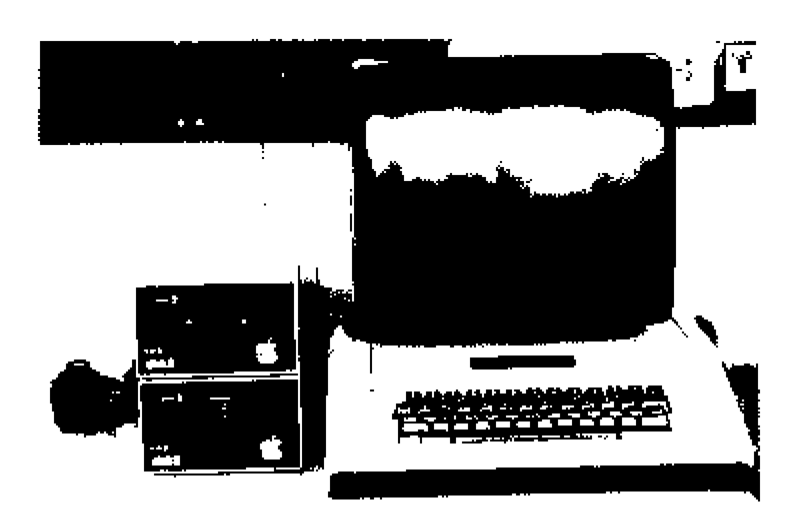


Bild 3.27: APPLE II mit Monitor und zwei Diskettenlaufwerken

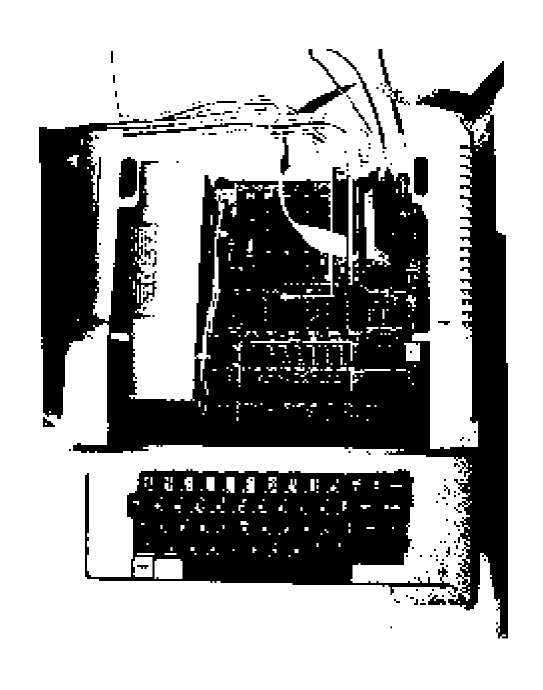


Bild 3.28: APPLE II, Blick in Gehäuseinneres

Derart nachgerüstet entspricht dann auch der APPLE II wieder unserer Beschreibung eines Standard-6502-Mikrocomputers.

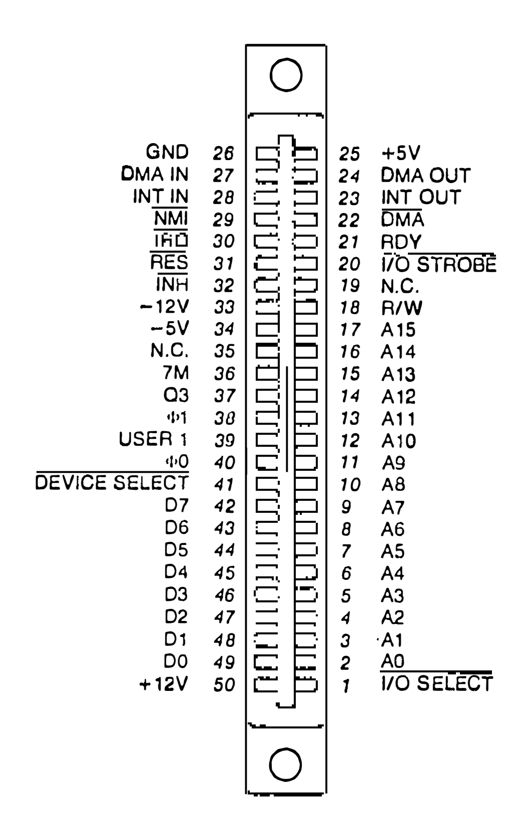


Bild 3.29: Periphere Ein/Ausgabestecker bei APPLE II (slots), Kontaktbelegung

Table 23: Peripheral Card I/O Locations

	$0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$C080										0						
$C090										1						
$C0A0										2						
$C0B0										3						
$C0C0	Ein-/Ausgabeadresse für slot-Nummer									4						
$C0D0										5						
$C0E0										6						
$C0F0										7						

Bild 3.30: APPLE II, periphere Ein/Ausgabe-Stecker, Adressen

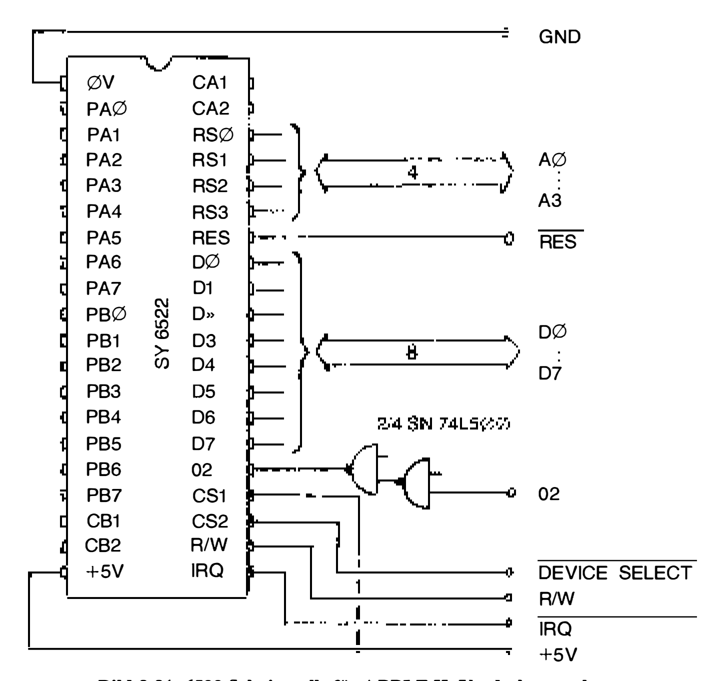


Bild 3.31: 6522-Schnittstelle für APPLE II, Verdrahtungsplan

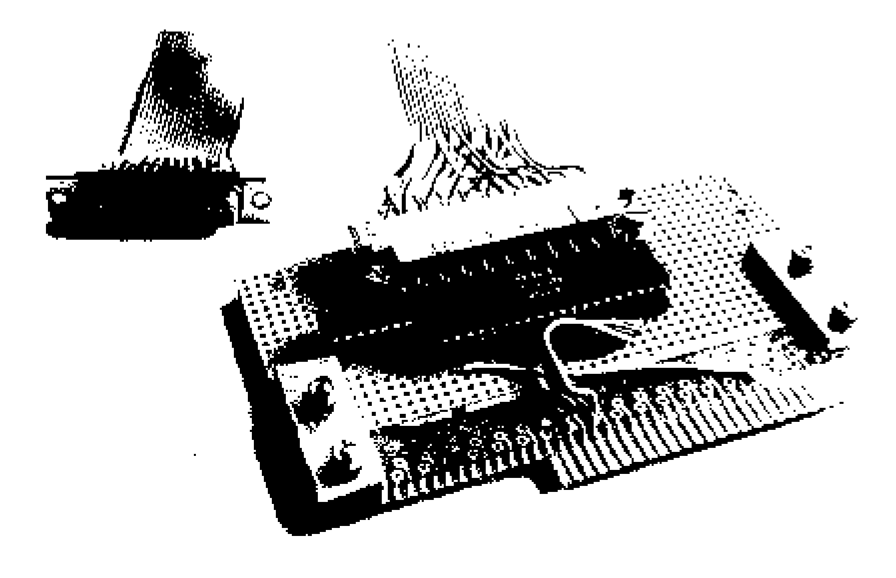


Bild 3.32: 6522-Schnittstelle für APPLE II, Probeaufbau

Weitere Mikrocomputer

Weitere Mikrocomputer werden von verschiedenen Herstellern, wie etwa Ohio Scientific, produziert.

Im Großen und Ganzen entsprechen alle diese 6502-Mikrocomputer mit gewissen Einschränkungen der Beschreibung unseres „Standard-Systems". Solange sie dieselben E/A-Bausteine verwenden (und das ist fast immer der Fall, da diese sehr entscheidende Vorteile bieten), sollten bei den Programmen in diesem Buch praktisch keine Änderungen nötig sein, außer eventuell bei PIO-Adressen oder falls bestimmte E/A-Leitungen nicht verfügbar sind,Die Steckerleisten A und E des SYM entsprechen den Steckerleisten bei KIM und AIM. Die senkrecht stehende Platine, links von dem Netzteil in Bild 3.31, ist eine 16K-Speicherbereichs-Erweiterungsplatine, die über die E-Steckerleiste angeschlossen ist.

Im Vordergrund sehen Sie zwei Versuchschaltungen, die an die A-Stekkerleiste angeschlossen sind: eine hexadezimale Tastatur und ein Minidrucker. Eine Beschreibung finden Sie in Kapitel 6.



Bild 3.31: KIM/SYM/AIM Anschluß-Kompatibilität

Kapitel 4

Standardtechniken

Einleitung

Wir werden in diesem Kapitel einen 6502 Mikrocomputer an Standard-E/A-Bausteine anschließen. Wir werden ihn an einfache Ausgabeeinheiten wie LED's (lichtemittierende Dioden), Relais oder Lautsprecher anschließen. Mit einem Satz Schalter werden wir Eingabeoperationen durchführen. Mit Hilfe dieser Zusatzgeräte werden wir dann beginnen, einfache Anwenderprogramme zu entwickeln: einen Morsegenerator, eine 24-Stunden-Uhr, ein einfaches Heim-Steuerprogramm und sogar einen automatischen Telefonwähler. Danach werden wir direkte Anwendungen dieser Techniken vorstellen: eine Sirene, ein Pulsmeßgerät, ein Musikprogramm und ein mathematisches Spiel. Im folgenden Kapitel werden wir dann etwas komplexere Programme entwickeln, für die wir außer diesen Standard-E/A-Bausteinen auch komplexere benutzen werden.

Für den Aufbau der in diesem Kapitel beschriebenen Anwenderplatine werden nur wenige Bausteine benötigt. Ein Bild dieser Platine sehen Sie in Bild 4.0. Alle Bauteile können Sie billig in jedem Elektronikgeschäft kaufen. Dem Leser sei dringend empfohlen, sich diese wenigen Bauteile zu beschaffen und wie im folgenden Kapitel beschrieben zusammenzuschalten, um die hier angegebenen Programme auch wirklich ausprobieren zu können. Die Grundvoraussetzung ist natürlich das Vorhandensein eines 6502 Mikrocomputers.

Wir werden bei der Vorstellung der Programme im ersten Teil die Hardwarekonfiguration des SYM, im zweiten die des KIM verwenden. Jedoch sollten alle diese Programme mit geringfügigen Änderungen auf beliebigen anderen 6502-Computern laufen (siehe auch Kapitel 2).

Die in diesem Kapitel entwickelten Programme sind einfach, setzen jedoch ein grundlegendes Verständnis des Befehlssatzes des 6502 voraus. Dieser wird im ersten Buch dieser Reihe, „Programmierung des 6502“ vorgestellt.

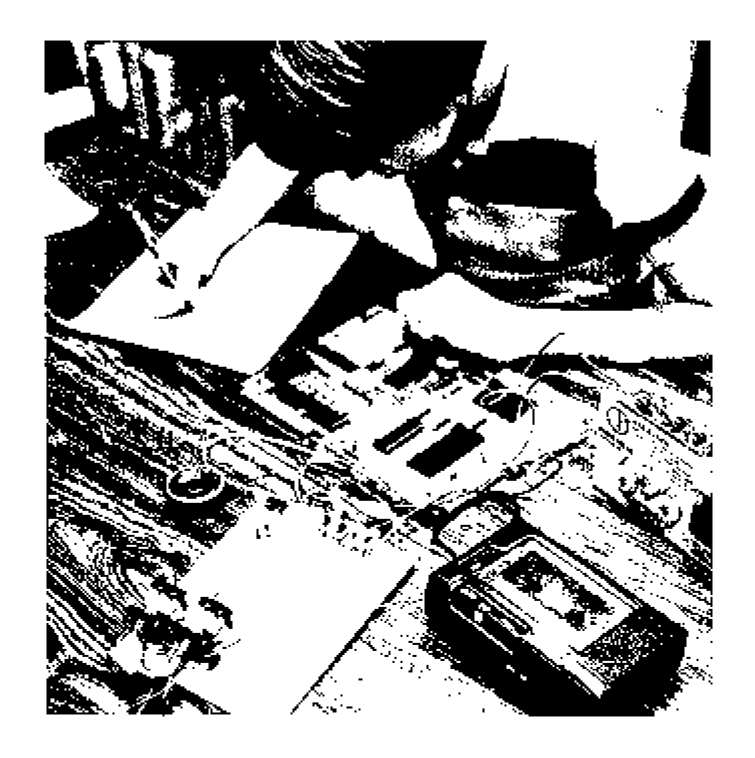
Hier die Liste der für die Anwenderprogramme dieses Kapitels benötigten Bauteile:

1 gelochte Platine (z. B. Veroboard)
4 Schalter (1 x UM)
1 LED-Treiber
mehrere LED's
3 Relais (12 V, 50..500 Ohm)
1 hochohmiger Lautsprecher
1 Potentiometer (1 KOhm..10 KOhm, linear)
1 Netzstecker für 220 V
2 Netzkupplungen für 220 V
verschiedene Widerstände

Wie man die verschiedenen Bauteile untereinander verbindet, wird bei den jeweiligen Beispielen erläutert.

Der Zusammenbau der Anwenderplatine ist zum Verständnis dieses Kapitels nicht absolut unverzichtbar. In diesem und den folgenden Kapiteln werden jedoch viele Übungsaufgaben gestellt werden. Obwohl man sie auch auf dem Papier lösen kann, erlangt man das richtige Programmierverständnis am besten durch einfaches Ausprobieren. Wir wollen dem Leser daher nochmals ans Herz legen, mit dem Programmieren entweder vor oder direkt nach dem Lesen dieses Buches an einem geeigneten Computer zu beginnen.

Es ist das Ziel dieses Kapitels, die Standardtechniken von Hardware- und Softwareinterfaces, die man braucht, um einfache externe Geräte an irgendeine 6502-Platine anzuschließen, zu erklären. Sie sollten am Ende dieses Kapitels in der Lage sein, die wichtigsten Möglichkeiten der Ein-/Ausgabebausteine auszuschöpfen und Programme zur Abfrage und Steuerung von E/A-Bausteinen selbst zu schreiben. Im nächsten Kapitel werden Sie auf diesem Wissen aufbauen und komplexere Heim- und Industrieanwendungen entwickeln.



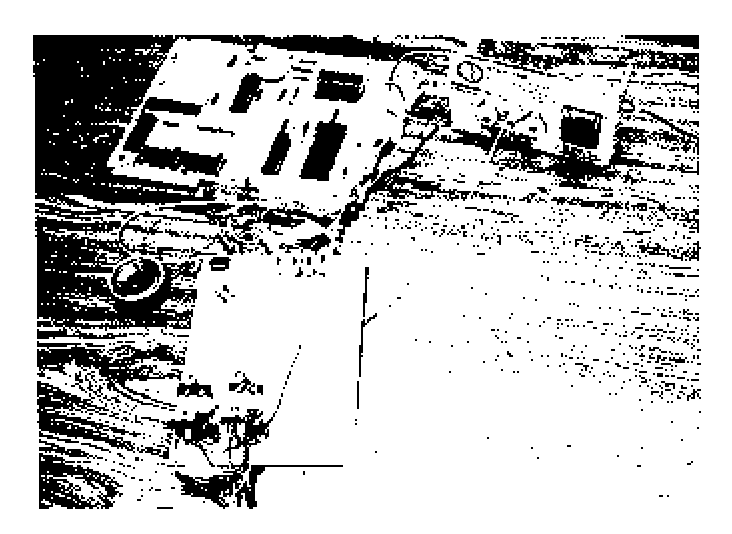


Bild 4.0: Ein vollständiges 6502-System: Stromversorgung, Mikrocomputerlatine, Kassettenrecorder und Anwenderplatine

Abschnitt 1: Die Grundtechniken

Relais

Ein Relais wird zur Steuerung einer äußeren hohen Spannung oder von Starkstromschaltungen eingesetzt: das Relais *trennt* den Steuerkreis von dem Externen. Ein Relais benötigt Gleichstrom. Dieser Strom fließt durch eine Spule und erzeugt ein magnetisches Feld. Durch dieses Feld wird wiederum ein Anker angezogen, der einen beweglichen Kontakt schließt. Der *äußere Stromkreis* kann sowohl Gleich- als auch Wechselstrom führen. Zur Steuerung externer Geräte, die hohe Ströme oder hohe Spannungen benötigen, wie etwa Elektrogeräte, verwenden wir Relais.

Der SYM Mikrocomputer trägt auf seiner Platine eine spezielle Einrichtung für Geräte mit höherer Spannung oder höherem Strom. Auf der Platine sind vier gepufferte Ausgänge verfügbar. Sie sind an die Bits 4, 5, 6 und 7 des Ein-/Ausgaberegisters B des PIO #3 (6522-U29) angeschlossen (siehe Bild 4.1). Daher werden wir direkt diese vier Ausgänge benutzen. Sie sind in der Lage, direkt Relais zu steuern. Für alle anderen Mikrocomputerplatinen, die nur ungepufferte PIO-Ausgänge haben (z. B. CBM), sind Treibertransistoren oder Puffer nötig. Die Verwendung von Treibertransistoren zeigt Bild 4.2. Von zwei Ausgangsleitungen eines PIO werden hier zwei externe Relais gesteuert. Für die Treibertransistoren eignet sich praktisch jeder NPN-Typ (z. B. BC 109). Statt an Tor B lassen sich die Relais natürlich auch an Tor A anschließen (z. B. bei CBM/VC20). Lediglich die Speicheradressen sind dann entsprechend abzuändern.

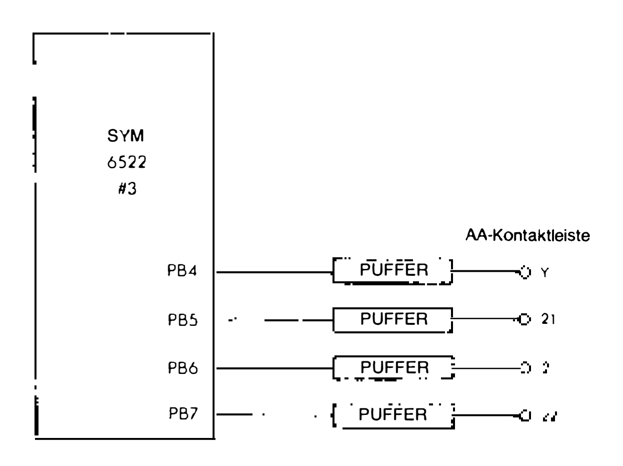


Bild 4.1: E/A-Puffer

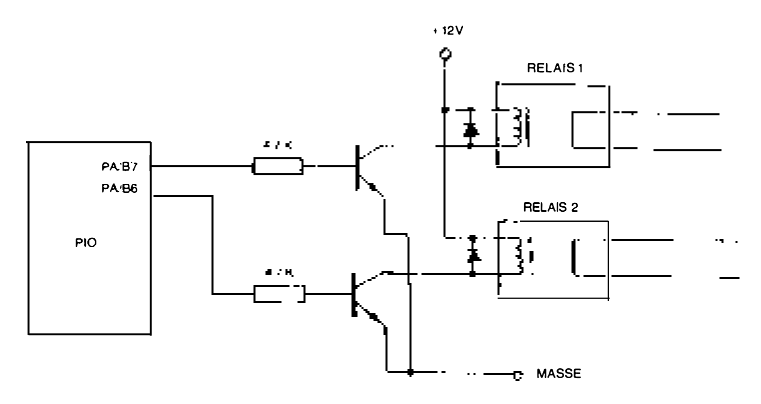


Bild 4.2: Relais-Interface

Das Hardware-Interface

Das Schaltbild für ein einzelnes Relais erscheint in Bild 4.3. Dieses Relais kann beispielsweise ein 12V Relais mit einem Spulenwiderstand von 100 bis 500 Ohm sein. Bei dem Kontakt kann es sich um einen einzelnen Arbeitskontakt oder um einen Umschaltkontakt jeweils mit einer Strombelastbarkeit von 6 bis 10A handeln. Die Strombelastbarkeit der Relaiskontakte sollte hoch genug gewählt werden, so daß sie das angeschlossene externe Gerät auch sicher schalten können. Die meisten Haushaltsgeräte ziehen nicht mehr als 6 bis 10A, so daß der oben gegebene Richtwert für Anwendungen im Haushalt ausreichen sollte.

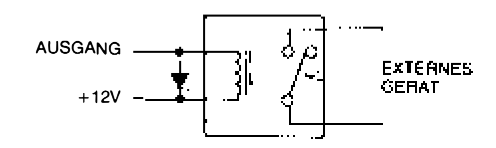


Bild 4.3: Anschluß eines einfachen Relais (Arbeitskontakt)

Beachten Sie bei diesem Bild, daß parallel zur Relaisspule eine Schutzdiode in Sperrichtung geschaltet ist. Dies ist eine wichtige Schutzmaßnahme bei jedem Relais, die eine Zerstörung des PIO-Puffers oder Treibers verhindert. Wenn das Relais abgeschaltet wird, tritt eine hohe Spannungsspitze umgekehrter Polarität an der Relaisspule auf. Es kann jede Diode ausreichender Spannungsfestigkeit eingesetzt werden. Für unsere Belange ist beispielsweise eine 1N914, eine 1N4148 oder eine 1N4001 ausreichend.

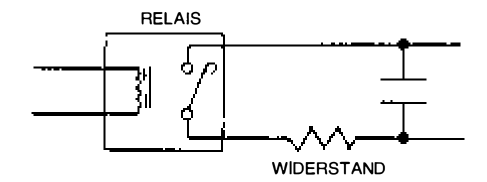


Bild 4.4: Schutzmaßnahmen auf der Geräteseite

Auf der Geräteseite des Relais können wir folgende Vorkehrungen treffen: parallel zum Ausgang kann ein Kondensator geschaltet werden, um Spitzen beim Öffnen der Kontakte kurzzuschließen (dadurch steigt die Lebensdauer der Relaiskontakte); ferner sollte ein Serienwiderstand eingefügt werden, falls hohe Einschaltströme auftreten können (Bild 4.4).

Ein Relais mit einem Doppelkontakt wird genauso angeschlossen, Bild 4.5 zeigt den Schaltplan. Solch ein Relais ist in der Lage, gleichzeitig zwei voneinander unabhängige Geräte zu schalten.

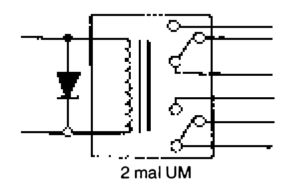


Bild 4.5: Anschluß eines Relais mit 2 Kontakten

Wir wollen nun eine praktische Anwendung betrachten. An die Bits 6 und 7 von Tor B des SYM-PIO werden wir zwei Relais R1 und R2 anschließen. Diese Relais werden wir zur Steuerung von Wechselstromverbrauchern einsetzen. Wir wollen im einfachsten Fall annehmen, daß diese Verbraucher zwei getrennte Lampen sind. Das wird es uns erlauben, das Programm ganz einfach auszuprobieren, indem wir uns ansehen, ob die Lampen korrekt ein- und ausgeschaltet werden. Natürlich könnte man anstelle der Lampen auch jedes andere Haushaltsgerät oder jeden anderen Verbraucher, der das Relais nicht überlastet, anschließen. In Bild 4.6 zeigen wir den Schaltplan.

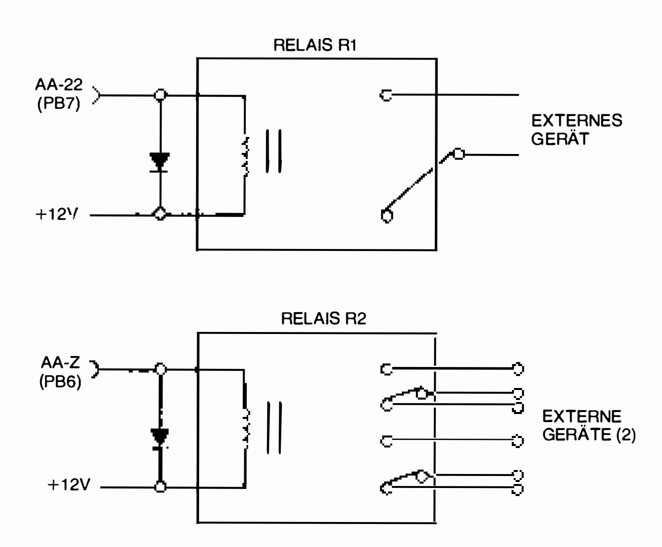


Bild 4.6: Anschluß zweier Relais an den PIO

Wir wollen die drei Bilder 3.11, 3.12 und 3.13, die die Anschlußbelegung der drei Kontaktleisten des SYM zeigen, untersuchen: wir sehen, daß die vier gepufferten Ausgänge PB4, PB5, PB6 und PB7 an den Kontakten Y, 21, Z und 22 zur Verfügung stehen. Die Verbindungspunkte, die wir im Schaltbild mit PB6 und PB7 bezeichnen, brauchen daher nur mit kurzen Drahtstücken an die entsprechenden Kontakte des „Hilfs-Anwendersteckers“ (AA) angeschlossen zu werden.

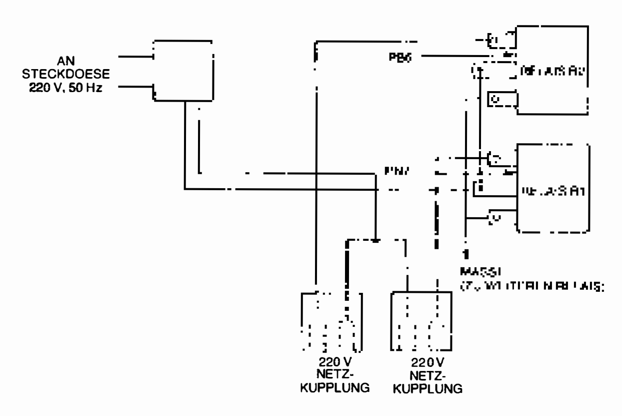


Bild 4.7: Externe Beschaltung der Relais

Auf der zum externen Gerät hin zeigenden Seite des Relais wird ein Netzstecker angeschlossen, der vom Netz 220V liefert. Die beiden Netzkupplungen, die vom Mikrocomputer gesteuert werden, erhalten von hier ihre Spannung. Diese beiden Kupplungen werden nach Bild 4.7 an die Relais angeschlossen. Jede einzelne kann aber unabhängig von der anderen vom Mikrocomputer ein- oder ausgeschaltet werden. Wir wollen nun die Softwaresteuerung für diese Relais erstellen.

Adresse	Register
AC00	IORB
AC02	DDRB
AC05	T1L-H
AC06	T1L-L
AC07	T1C-H
AC0B	ACR
AC0F	

Bild 4.8: Speicherbelegung für 6522 #3 (der dritte 6522 beim SYM)

Das Software-Interface

Jedes der zwei an die Relais R1 und R2 angeschlossenen Geräte wird eingeschaltet, wenn das entsprechende Relais anzieht. Das Relais zieht an, sobald das entsprechende Ausgangsbit auf „1“ gesetzt wird. Mit einem Blick auf Bild 4.8 sieht man, daß sich das Tor B für den 6522 #3 an der Speicheradresse AC00 befindet. Den Inhalt des Speichers AC00 zeigt Bild 4.9. Schalten wir die Relais jetzt also ein und aus.

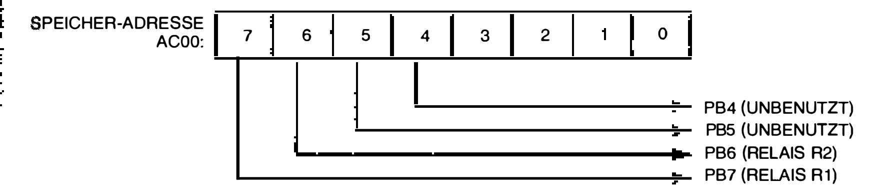


Bild 4.9: Tor B des 6522 #3

Als erstes müssen wir Tor B als Ausgabetor vereinbaren. Um es uns einfach zu machen, werden wir alle Bits von 0 bis 7 als Ausgabekanäle definieren, obwohl wir nur die Bits 6 und 7 brauchen werden. Für andere Anwendungsfälle läßt sich diese Vereinbarung leicht abändern. Von Kapitel 2 wissen wir, daß wir die Bits des Datenrichtungsregisters mit einer Null oder einer Eins laden müssen, um festzulegen, in welcher Richtung die entsprechende E/A-Leitung betrieben werden soll. Eine Eins im Datenrichtungsregister legt die Ausgabefunktion fest. Eine Null legt die Eingabefunktion fest. Wenn wir lauter Einsen in das Datenrichtungsregister laden, werden alle Bits als Ausgabeleitungen definiert.

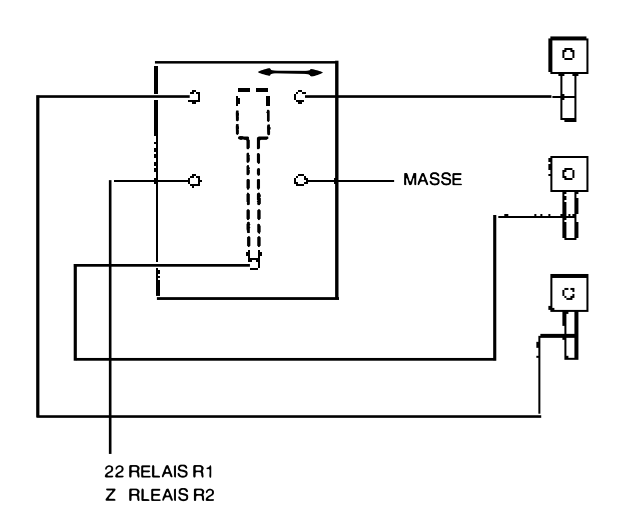


Bild 4.10: Detail des Relaisanschlusses an die Anwenderplatine

An dieser Stelle ist folgender Hinweis angebracht: es ist stets von Vorteil, wenn man beim Programmieren so vorgeht, daß man alle Probleme so einfach und konsistent wie möglich löst. Da wir in diesem Beispiel vorläufig annehmen, daß an die anderen Ausgänge von Tor B keine weiteren

Geräte angeschlossen sind, ist es am sichersten, einfach alle Leitungen als Eingabekanäle bzw. als Ausgabekanäle zu definieren.

Mit der folgenden Befehlsfolge legen wir alle Bits als Ausgabe fest:

```
LDA  #$FF      LADE A MIT „11111111“
STA  $AC02     LEGE A IM SPEICHER
               AC02 AB
```

Anhand von Bild 4.8 kann man sich überzeugen, daß AC02 in der Tat die Adresse des Datenrichtungsregisters von Tor B des 6522 #3 ist. Hexadezimal „FF“ entspricht binär „11111111“. Nun wollen wir das an PB6 angeschlossene Relais R2 einschalten.

```
LDA  $AC00     LIES AKTUELLEN WERT
               VON TOR B
ORA  #$40      SETZE PB6 AUF 1
STA  $AC00     AUSGABE
```

Mit dem ersten Befehl liest man den aktuellen Wert des Tores B ein. Da etliche andere Geräte oder Relais an Tor B angeschlossen sein können, wollen wir nicht einfach die Zahl „01000000“ nach IORB schreiben; das würde zwar das an PB6 angeschlossene Relais einschalten, gleichzeitig aber alle anderen Relais abschalten! Wir wollen daher den aktuellen Wert von PB einlesen und nur ein einzelnes Bit ändern, PB6. Diese Änderung machen wir mit der logischen ODER-Verknüpfung, dem zweiten Befehl in diesem Programmteil („ORA“). Die logische ODER-Verknüpfung läßt alle anderen Bits unverändert und erzwingt bei diesem einen festgelegten Bit eine Eins. Falls wir PB7 anstatt PB6 einschalten wollten, würde für die Verknüpfung die Zahl $80 hexadezimal (= „10000000“ binär) anstelle von $40 verwendet werden. Schließlich wird das resultierende Bitmuster in die Adresse AC00 abgelegt (diese Adresse entspricht dem Tor B). Das an PB6 angeschlossene Relais wird hierdurch eingeschaltet.

Übungsaufgabe 4.1: *Schreiben Sie mit drei Befehlen ein Programm, das die an PB6 und PB7 angeschlossenen Relais R1 und R2 gleichzeitig einschaltet.*

Nun wollen wir das an PB6 angeschlossene Relais R2 wieder ausschalten:

```
LDA  $AC00     LIES AKTUELLEN WERT
               VON TOR B
AND  #$BF      SETZE BIT 6 AUF NULL
STA  $AC00     SPEICHERE ERGEBNIS
               NACH PB
```

Die logische UND-Verknüpfung wird hier eingesetzt, um das angegebene Bit auf Null zu setzen. Alle anderen Bits werden nicht verändert („BF“ hexadezimal entspricht „10111111“ binär).

Hinweis: Üblicherweise verwendet man die UND-Verknüpfung, um bestimmte Bits auf Null zu setzen. Dasselbe Ergebnis läßt sich jedoch auch mit der EOR-Instruktion erzielen. Das Programm bleibt unverändert, nur aus der Zeile „AND #$BF“ wird

EOR #$40

Der Vorteil ist, daß der zum Ausschalten verwendete Befehl derselbe ist, wie der zum Einschalten verwendete. Dadurch werden mögliche Fehler vermieden. Natürlich sollte sich der Leser überzeugen, daß diese Methode tatsächlich eine Null erzwingt. Der Grund ist, daß die exklusive Oder-Verknüpfung von „1“ mit „1“ eine „0“ ergibt. Falls Bit 6 auf „1“ gesetzt war, wird EOR #$40 es also zu Null machen. Alle anderen Bits bleiben unberührt.

Verifizierung

Wir wollen nun ausproblieren, ob diese einfache Befehlsfolge tatsächlich ausreicht, die Relais ein- und auszuschalten. Wir werden zwei Lampen oder andere Geräte an die beiden Relais anschließen, unser Programm über die Tastatur eingeben und dann nachsehen, ob die beiden Lampen ein- oder ausgeschaltet werden. Da die Eingabe über die Tastatur in hexadezimaler Form erfolgen muß, zeigen wir hier auch das hexadezimale Äquivalent unserer beiden Programme:

Zum Einschalten des Relais:

```
AD  00  AC
09   MUSTER    (MUSTER STEHT FÜR
                8-BIT-MUSTER)
8D  00  AC
```

Und zum Ausschalten:

```
AD  00  AC
29 bzw. 49 Muster  (AND# = $29,
                    EOR# = $49)
8D  00  AC
```

Falls Sie einen Mikrocomputer besitzen, geben Sie nun die beiden Programme ein und probieren Sie deren korrekten Ablauf aus.

Schalter

Zwei Sorten von Schaltern können angeschlossen werden: Einschalter und Umschalter. Bild 4.11 zeigt, wie ein Einschalter (1 x Ein) angeschlossen wird. Mit der hier gezeigten Verdrahtung ist der Schalter auf logisch „1“, wenn der Kontakt offen ist und auf logisch „0“ bei geschlossenem Kontakt. Will man es umgekehrt haben, müssen lediglich die Polaritäten am Schalter gewechselt werden.

Bild 4.12 zeigt Ihnen, wie ein Umschalter (1 x Um) anzuschließen ist. Die Verdrahtung ist ganz direkt. Eine der Schaltstellungen ist logisch „1", die andere entsprechend logisch „0".

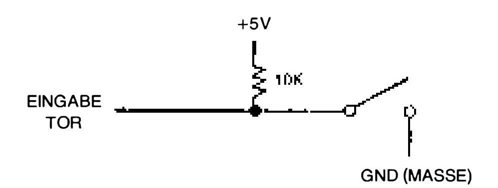


Bild 4.11: Anschluß eines Einschalters

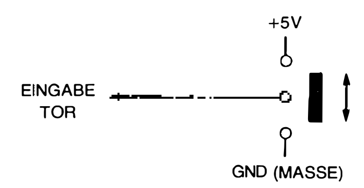


Bild 4.12: Anschluß eines Umschalters

Anschluß von vier Schaltern

Wir werden die Leitungen 1, 2, 3 und 4 von Tor B des 6522 #1 als vier Eingabeleitungen verwenden und mit ihnen den Status der externen Schalter abfragen. Den Schaltplan sehen Sie in Bild 4.13. Wir wollen das zugehörige Programm untersuchen.

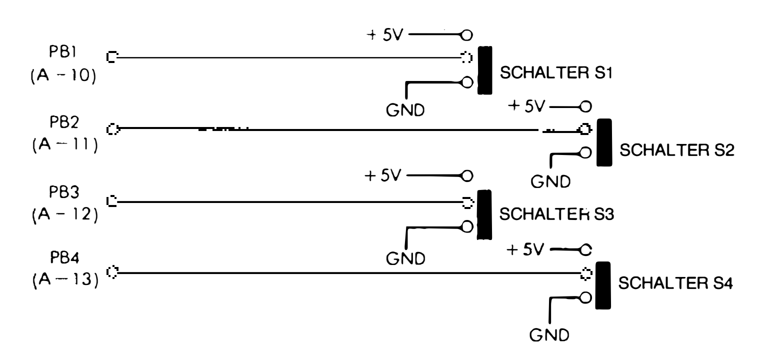


Bild 4.13: Anschluß von vier Umschaltern an den SYM

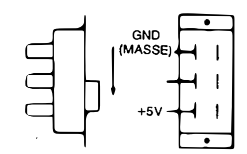


Bild 4.14: Ein Umschalter (Hier: Schiebeschalter, 2 x UM)

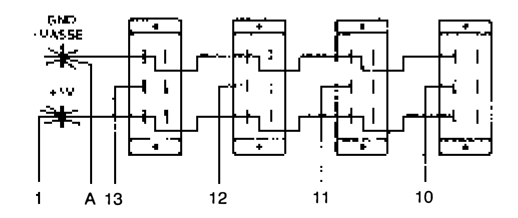


Bild 4.15: Anschlußdetails bei vier Umschaltern

Das Software-Interface

Als erstes müssen wir PB1, PB2, PB3 und PB4 als Eingabeleitungen von Tor B vereinbaren. Hierzu laden wir in den Speicher mit der Adresse A002 das geeignete Bitmuster. A002 ist die Adresse des Datenrichtungsregisters des Tores B des 6522#1.

```
LDA  #$E0      SETZE BITS 01234 AUF
               EINGABE
STA  $A002
```

Das Bitmuster „EO“ legt die Leitungen 0, 1, 2, 3, 4 als Eingabekanäle und die Leitungen 5, 6, 7 als Ausgabekanäle fest (letztere können mit externen Relais verbunden sein). „EO“ hexadezimal ist „11100000“ binär. Jede „0“ legt die Eingabefunktion, jede „1“ die Ausgabefunktion fest. Da Bit 0 eigentlich unbenutzt bleibt, könnte man auch „E1“ verwenden.

Wir wollen nun den Wert der Schalter einlesen und an eine je nach eingelesenem Wert festgelegte Speicheradresse springen.

```
LDA  #SCHALTER „02“ FÜR S1, „04“ FÜR S2,
               „08“ FÜR S3, „10“ FÜR S4
BIT  $A000     UND IORB
BEQ  SPEZADR   VERZWEIGUNG ZUR
               ANGEGEBENEN ADRESSE,
               WENN SCHALTER NULL
               (AUS) WAR
```

Falls wir zu der spezifizierten Speicheradresse verzweigen wollen, wenn der entsprechende Schalter auf „1" steht (=AN), müssen wir die Anweisung BEQ in der letzten Programmzeile durch BNE ersetzen.

Testen des Programmes auf dem Mikrocomputer

Die hexadezimale Kodefolge für das obige Programm sieht folgendermaßen aus:

	A9	SCHALTER	
	2C	00 A0	
	FO	SPEZADR	(FÜR BEQ)
bzw.:	DO	SPEZADR	(FÜR BNE)

Lautsprecher

Man kann einen externen Lautsprecher direkt an den Ausgang eines der PIO's anschließen. Anschluß 7 ist meist leistungsfähiger und wird daher i. a. verwendet. Beim 6522 kann das Ausgangssignal an PB7 durch einen der internen Zeitgeber gesteuert werden. Der Zeitgeber wird zur Erzeugung eines Tones vorgegebener Frequenz verwendet. Der bevorzugte Anschluß zum Betreiben eines Lautsprechers wird daher PB7 sein. Wie man ihn anschließt zeigt Bild 4.16.

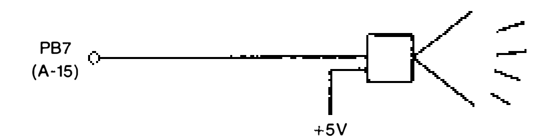


Bild 4.16: Anschließen des Lautsprechers

Wenn der gepufferte Ausgang des SYM (6522#3) verwendet wird, empfiehlt sich das Einfügen eines Serienwiderstandes in Reihe zum Lautsprecher zur Begrenzung des Ausgangsstromes. Anstatt den Lautsprecher direkt an einen PIO-Ausgang anzuschließen, kann man auch die Schaltung nach Bild 4.17 verwenden. Man erhält damit einen lauteren Ton.

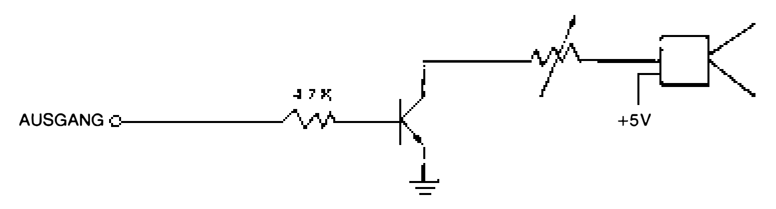


Bild 4.17: Erzielen eines lauteren Tones

Warnung: Das Potentiometer in Bild 4.17 dient der bequemen Einstellbarkeit der Lautstärke. Wird es jedoch ganz auf Null zurückgedreht, wird es sehr wahrscheinlich durchbrennen und den Treibertransistor zerstören (u. U. auch den des SYM). Man vermeidet dies durch Einfügen eines Festwiderstandes in Serie zum Potentiometer (etwa 100 Ohm) oder durch Verwendung eines hochohmigen Lautsprechers.

Das Software-Interface

Man kann mit dem Lautsprecher einen Ton erzeugen, indem man ihn mit der gewünschten Frequenz einfach ein- und ausschaltet. Ein solcher Ton wird nicht den „sauberen Klang" eines Musikinstrumentes erreichen, da wir ihn mit einem Rechtecksignal erzeugen. Für unsere Ansprüche wird das aber völlig ausreichen und wir können die Töne durch ihre Frequenz problemlos unterscheiden. Wir werden gleich zu einer praktischen Anwendung übergehen.

Ein Morsegenerator

Wir werden ein Programm entwickeln, das in der Lage ist, zu jedem Buchstaben des Alphabets den entsprechenden Morse-Kode zu erzeugen. Das Programm wird einen Lautsprecher steuern, so daß wir die einwandfreie Funktion des Morsegenerators überprüfen können. Zusätzlich kann es ein externes Gerät ein- und ausschalten, so daß das Zeichen im Morse-Kode beispielsweise von einem Sender ausgestrahlt werden kann.

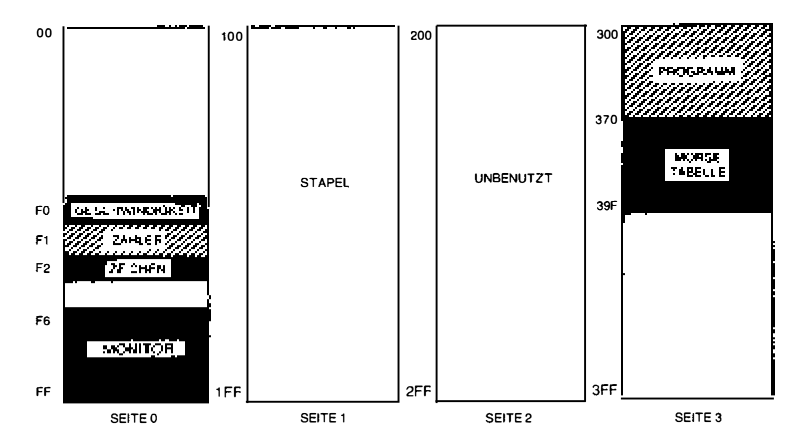


Bild 4.18: Speicherbelegung für das Morseprogramm

Für das Programm werden die folgenden Vereinbarungen getroffen:

Das Programm selbst wird auf Seite 3 des RAM abgelegt, startet also bei Adresse 0300. In Bild 4.18 ist dies dargestellt. Das Programm enthält auch eine Morsezeichentabelle, mit der für jedes vorgegebene ASCII-Zeichen das entsprechende Bitmuster des Morsezeichens erzeugt wird. Der Aufbau dieser Tabelle wird weiter unten erläutert. Wir nehmen an, daß das erste Zeichen, das in den Morsecode umgesetzt werden soll, schon im Akkumulator steht, wenn das Programm startet.

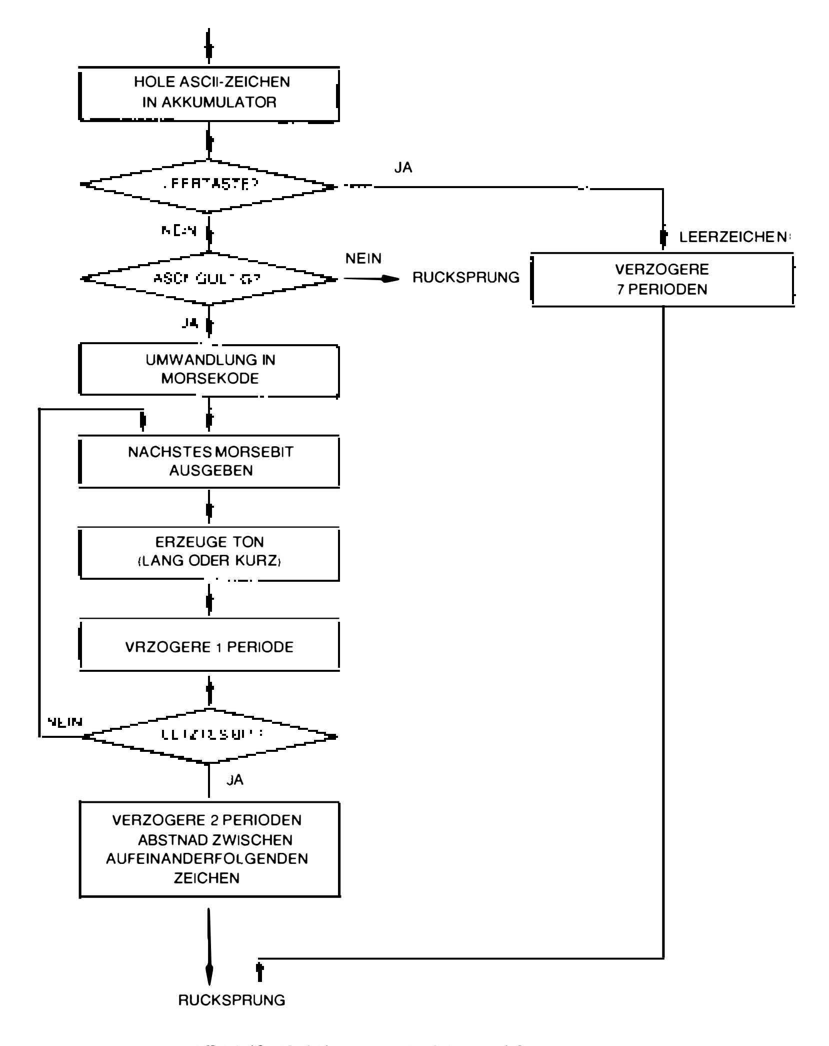


Bild 4.19: Flußdiagramm der Morsezeichenerzeugung

Ferner soll die Aussendegeschwindigkeit mit der Variablen ‚GESCHWINDIGKEIT', die auf der Seite 0 im Speicher F0 steht, einstellbar sein (siehe Bild 4.18). Die Zeiteinheit (das ist die Dauer eines Punktes im Morse-Kode) wird intern in Einheiten von Millisekunden ausgedrückt. Der Wert „100" in der Variablen ‚GESCHWINDIGKEIT' resultiert in Punktlängen von einer Zehntelsekunde.

Bevor man das Programm startet, sollen die Variablen ‚ZEICHEN' und ‚GESCHWINDIGKEIT' gültige Werte enthalten und das zu sendende Zeichen soll schon im Akkumulator stehen. Ein externes Unterprogramm könnte diese Routine wiederholt aufrufen, um ganze Zeichenketten auszusenden. Dieses Unterprogramm ist dann dafür verantwortlich, das nächste Zeichen jedesmal vor dem Aufrufen des Morsegenerators in den Akkumulator zu bringen.

Wir wollen nun den Algorithmus studieren, der zur Aussendung des Morsezeichens verwendet wird.

Dieser Algorithmus wird im Flußdiagramm im Bild 4.19 dargestellt. Als erstes prüft das Programm, ob ein Leerzeichen (Space) vorliegt. Falls ja, wird für die Zeit von 7 Perioden kein Signal erzeugt.

Dann überprüft es, ob das ASCII-Zeichen, das im Akkumulator steht, einen zulässigen hexadezimalen Wert hat. Zugelassene Zeichen liegen zwischen „2C" und „5A" einschließlich (hexadezimaler 7-Bit ASCII-Code). Andernfalls liegt ein Fehler vor und das Programm springt zurück. Nach der Feststellung der Gültigkeit eines Zeichens muß dessen ASCII-Code in den äquivalenten Morse-Kode umgesetzt werden. Diese Technik wird später erläutert.

Der Kode des Morsezeichens ist folgendermaßen aufgebaut: einem „START"-Bit (eine „1") folgt eine „0" für einen Punkt („·") und eine „1" für einen Strich („–"). Alle innerhalb des 8-Bit-Wortes nicht benutzten Bits links des Startbits werden auf „0" gesetzt. Die Umwandlung wird vom Programm mit einer später zu beschreibenden Tabellenaufruftechnik durchgeführt. Nehmen wir an, die binäre Version des Morsecodes läge schon vor. Eine Tonfolge muß dann erzeugt werden. Der Inhalt des Akkumulators wird nach links durchgeschoben, bis das Start-Bit gefunden wird. Nach der Erkennung des STARTbits wird jede „0" als ein „·" und jede „1" als ein „–" interpretiert, bis das achte Bit erreicht ist. Für jede durchgeschobene „0" wird ein „Kurz" erzeugt, für jede „1" ein „Lang". Auch die Tonerzeugung wird erst nachher im Detail erläutert.

Nach der Erzeugung des dem jeweiligen Bit entsprechenden Tones wird eine Warteschleife von einer Periode Dauer eingeschoben. Dann wird das nächste Bit des Morsezeichens geprüft, bis das letzte Bit (das achte) erreicht ist.

Nach Aussendung der Tonfolge, die dem Morsezeichen entspricht, wird eine Verzögerung von zwei Perioden erzeugt. Diese Verzögerung entspricht dem zwischen zwei Morsezeichen üblicherweise eingefügten Zwischenraum.

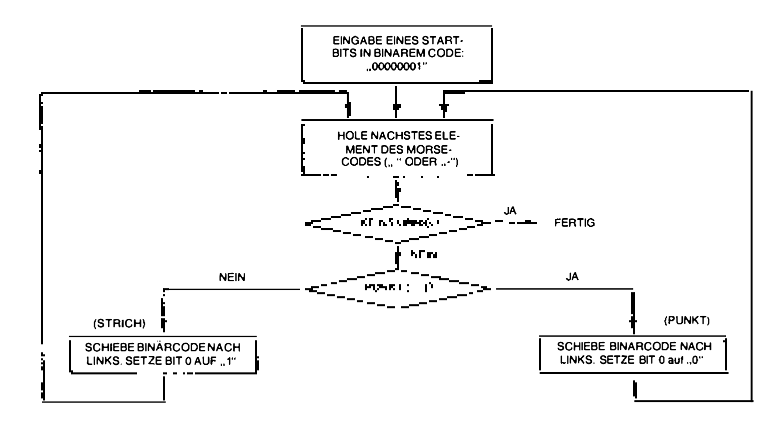


Bild 4.20: Umwandlung Morse nach binär

Diesen Ablauf erkennt man gut in Bild 4.19 im Flußdiagramm. Der Leser sollte sich von der Richtigkeit überzeugen. Nun wollen wir noch die speziellen Einzelprobleme studieren, die wir bisher noch nicht gelöst haben.

Umwandlung von ASCII in binären Morse-Kode

Wir wollen eine Umwandlungstabelle erstellen, die dem ASCII-Zeichen die binäre Darstellung seines entsprechenden Morsezeichens zuweist. Hierzu ein Beispiel: Das Zeichen „B“ hat den Morsecode „—···“. Jeder Strich wird durch eine „1“ und jeder Punkt durch eine „0“ kodiert. Das binäre Gegenstück zu „—···“ ist also „1000“.

Entsprechend der von uns eingangs getroffenen Vereinbarung müssen wir dem eben erzeugten Code noch ein Startbit voranstellen. Bisher haben wir also den binären Code „11000“. Schließlich soll jeder binäre Morse-Kode aus 8 Bit bestehen. Die links des Startbits noch fehlenden Bits füllen wir daher mit Nullen auf. Damit erhalten wir den 8-Bit Binärcode „00011000“, oder hexadezimal: „18“. Die hexadezimale Darstellung des Buchstaben „B“ ist also „18“.

Um Ihnen ein Beispiel zu geben, zeigt die Tabelle unten (Bild 4.21) die hexadezimalen Darstellungen der Buchstaben A, B, C, D. Bild 4.22 zeigt eine vollständige Umwandlungstabelle für alle existierenden Morsezeichen. Den Algorithmus, der der eben beschriebenen Umwandlung zugrunde liegt, erläutern die Bilder 4.20 und 4.23.

Buchstabe	ASCII	Morse	Binär	Hexadezimal
A	41	. —	00000101	05
B	42	— . . .	00011000	18
C	43	— . — .	00011010	1A
D	44	— . .	00001100	0C

Bild 4.21: Umwandlung ASCII in Morse

Zeichen	Morsecode	ASCII	Hexa-dezimal
,	— — . . — —	2C	73
—	— . . . . —	2D	31
.	. — . — . —	2E	55
/	— . . — .	2F	32
Ø	— — — — —	3Ø	3F
1	. — — — —	31	2F
2	. . — — —	32	27
3	. . . — —	33	23
4	. . . . —	34	21
5	. . . . .	35	2Ø
6	—	36	3Ø
7	— — . . .	37	38
8	— — — . .	38	3C
9	— — — — .	39	3E
:	Unbestimmt	3A	Ø1
;	" "	3B	Ø1
<	" "	3C	Ø1
=	" "	3D	Ø1
>	" "	3E	Ø1
?	. . — — . .	3F	4C
@	Unbestimmt	4Ø	Ø1
A	. —	41	Ø5
B	— . . .	42	18
C	— . — .	43	1A
D	— . .	44	ØC
E	.	45	Ø2
F	. . — .	46	12
G	— — .	47	ØE
H	. . . .	48	1Ø
I	. .	49	Ø4
J	. — — —	4A	17
K	— . —	4B	ØD
L	. — . .	4C	14
M	— —	4D	Ø7
N	— .	4E	Ø6
O	— — —	4F	ØF
P	. — — .	50	16
Q	— — . —	51	1D
R	. — .	52	ØA
S	. . .	53	Ø8
T	—	54	Ø3
U	. . —	55	Ø9
V	. . . —	56	11
W	. — —	57	ØB
X	— . . —	58	19
Y	— . — —	59	1B
Z	— — . .	5A	1C

Bild 4.22: Vollständige Morsetabelle

Wir haben nun die Umwandlungstabelle für alle ASCII-Zeichen erstellt. Wir werden diese Tabelle „Morsetabelle“ nennen und sie am Ende des Programmes abspeichern (Bild 4.18). Wenn wir die Darstellung eines bestimmten Zeichens benötigen, werden wir an der betreffenden Stelle in der Morsetabelle nachsehen und die binäre Darstellung finden. Diesen Zugriff beschreiben wir später, wenn wir das eigentliche Programm besprechen.

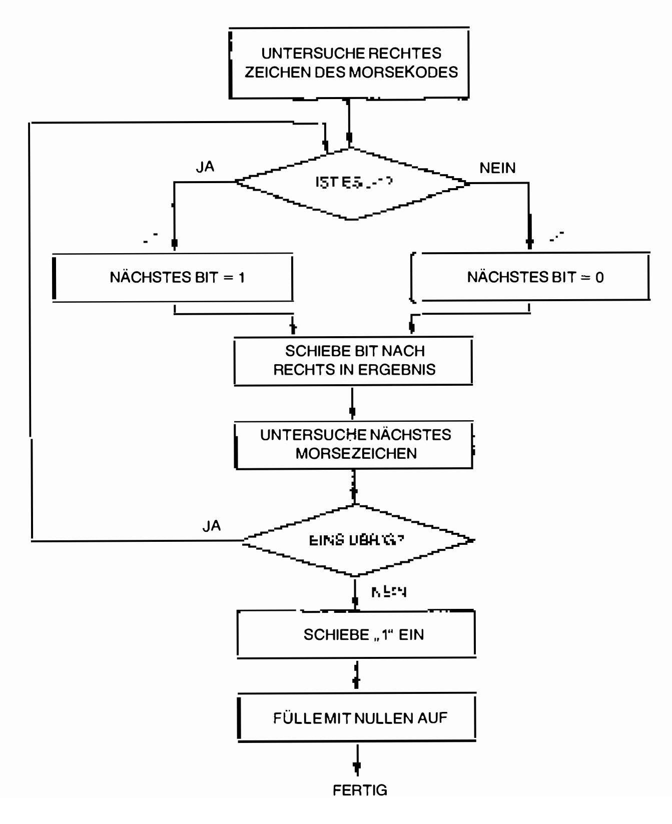


Bild 4.23: Flußdiagramm zur Erzeugung hexadezimaler Zeichen aus Morsezeichen

Tonerzeugung mit dem Zeitgeber

Unser nächstes Problem wird es sein, einen Ton vorgegebener Dauer und Frequenz zu erzeugen. Wir werden hierzu einen Zeitgeber verwenden.

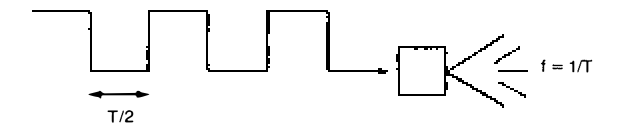


Bild 4.24: Rechtecksignal erzeugt Ton im Lautsprecher

Der Ton wird im Lautsprecher erzeugt, indem man ein Rechtecksignal der erwünschten Frequenz an ihn anlegt. Dies zeigt Bild 4.24. Man kann den Zeitgeber dazu verwenden, dieses Ausgangssignal automatisch zu erzeugen. Hierzu werden wir die entsprechenden Bits des ACR setzen (Bild 4.25) und dann einfach die Länge des erzeugten Tones steuern. Den genauen Zeitablauf zeigt Bild 4.26. Ø2 ganz oben im Bild ist die Phase 2 des System-Taktgenerators. In den meisten 6502-Systemen hat der Taktgenerator eine Periode von 1 µsec. Der von diesem Zeitgeber erzeugte Impuls erscheint am Ausgang PB7. Er hat eine Länge von N+1.5 Zyklen, wobei N der ins Zählregister geladene Wert ist. Das kommt daher, daß der Zähler von N nach O dekrementiert und dann den Ausgang bei der nächsten negativen Flanke des Taktgenerators invertiert. Diese Verhältnisse werden in Bild 4.26 wiedergegeben. Außerdem wird gleichzeitig ein Interrupt (IRQ) erzeugt, den wir hier aber nicht verwenden. Wir müssen also, um den Zeitgeber zu benutzen, dessen Zählregister mit einem entsprechenden Wert für N laden. Sobald wir jedoch diesen Zähler geladen

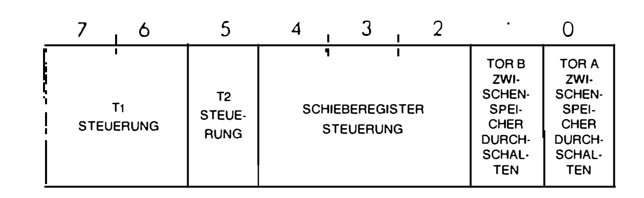


Bild 4.25: 6522 Hilfssteuerregister ACR

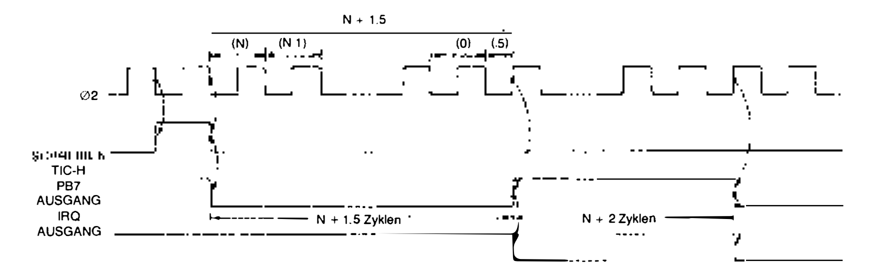


Bild 4.26: Diagramm der zeitlichen Abfolge bei der Tonerzeugung

haben, läuft der Zähler los. Weil das Zählregister ein 16 Bit Register ist, können wir es vom Mikroprozessor nicht mit einem einzigen Datentransfer laden lassen. Der Wert muß also *zwischengespeichert* werden. Daher ist der Zeitgeber mit einem internen 16 Bit Zwischenspeicher ausgestattet, der T1L heißt (L = Latch, d. h. Zwischenspeicher). Das niederwertige Byte des Zwischenspeichers heißt T1L-L (Latch-Low), das höherwertige T1L-H (Latch-High). Der Wert N wird nach T1L-L und T1L-H geladen. An dieser Stelle ist der 16-Bit-Inhalt eingegeben, aber es passiert noch nichts. Um den Zeitgeber zu starten, werden wir ein spezielles Startsignal geben, durch das der Inhalt des Zwischenspeichers in den eigentlichen Zähler übertragen wird. Dies ist der Befehl „Schreibe T1C-H", der im Bild 4.27 in der vierten Zeile steht:

```
LDA  #WERTLO
STA  $A006     LADE ZWISCHENSPEICHER,
               NIEDERWERTIG
LDA  #WERTHI
STA  $A007     LADE ZWISCHENSPEICHER,
               HÖHERWERTIG
STA  $A005     TRANSFER UND START
```

Bild 4.27: Programm zur Verwendung von Zeitgeber 1

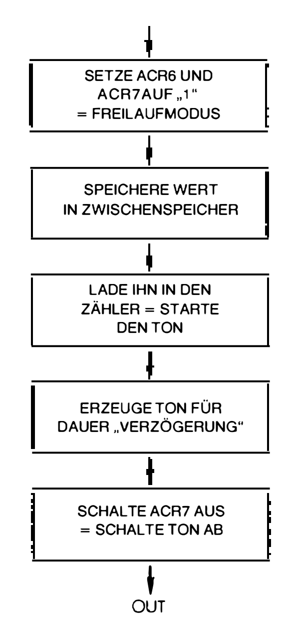
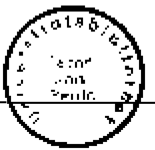


Bild 4.28: Erzeugen eines Tones bestimmter Dauer mit Zeitgeber 1



Die zum Einsatz des Zeitgebers nötige Befehlsfolge sollte damit klar sein. Sie wird im Flußdiagramm Bild 4.28 gezeigt. Als erstes werden wir die entsprechenden Bits des Steuerregisters ACR auf die richtigen Werte setzen. Der Zeitgeber arbeitet im Freilaufmodus, in dem er an PB7 ein fortlaufendes Rechtecksignal erzeugt. Man setzt hierfür die Bits 6 und 7 des ACR auf „0" und „1" (siehe Bilder 4.29 und 4.30). Als nächstes wird der Zwischenspeicher mit dem gewünschten Wert geladen. Er wird dann in das Zählregister selbst übertragen, um diesen zu starten. Von diesem Moment ab wird ein Ton erzeugt. Jedesmal, wenn der Zähler den Wert Null erreicht, wird er den im Zwischenspeicher befindlichen Wert wieder neu laden und somit automatisch ein Rechtecksignal mit einer halben Periodendauer von näherungsweise N+2 Taktzyklen erzeugen. (Dies ist deshalb nur eine Näherung, weil der Ausgangsimpuls für N+1,5 Taktzyklen auf 0 bleibt, während er für N+2 Taktzyklen auf 1 liegt.)

ACR7 STEUERUNG PB7	ACR6 STEUERUNG EINZEL/FREI	MODUS
0	0 (EINZEL-BETRIEB	Erzeugt Interrupt nach Ablauf der Zeit, mit der T1 geladen wurde. PB7 gesperrt.
0	1 (FREILAUF-MODUS)	Erzeugt Dauer-Interrupt. PB7 gesperrt.
1	0 (EINZEL-(BETRIEB)	Erzeug jedesmal, wenn T1 geladen wird, einen Interrupt und Ausgangssignal über PB7 = Einzelbetrieb mit programmierbarer Impulsbreite.
1	1 (FREILAUF-MODUS)	Erzeugt Dauer-Interrupt und Rechtecksignal auf PB7.

Bild 4.29: 6522 ACR legt Zeitgeber-Modus fest

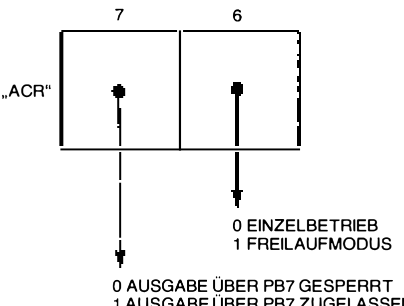


Bild 4.30: Die Bits 6 und 7 des ACR

```
ZEILE ADR      CODE        ZEILE
0002  0000                 ;DIESES UNTERPROGRAMM NIMMT ASCII-ZEICHEN ZWISCHEN 2CH UND
0003  0000                 ;SAH (SOWIE 20H FÜR SPACE) AN UND ERZEUGT DAS ENTSPRECHENDE
0004  0000                 ;MORSEZEICHEN MIT EINEM LAUTSPRECHER, DER AN PB7, 6522-U25
0005  0000                 ;ANGESCHLOSSEN IST. ES SCHALTET AUSSERDEM PBO VON 6522-U25
0006  0000                 ;EIN UND AUS. MIT EINEM PASSENDEN TREIBER KANN DIESES BIT
0007  0000                 ;EINEN SENDER TASTEN. EIN HAUPTPROGRAMM RUFT DIESES UNTER-
0008  0000                 ;PROGRAMM MIT DEM ZU SENDENDEN ZEICHEN IM AKKUMULATOR AUF.
0009  0000                 ;DAS HAUPTPROGRAMM KÖNNTE Z.B. VON DER TASTATUR EIN ZEICHEN
0010  0000                 ;EINLESEN UND MIT DIESEM PROGRAMM DEN MORSECODE AUSSENDEN,
0011  0000                 ;ODER MIT EINEM ZUFALLSZAHLENGENERATOR EINE REIHE ZEICHEN ER-
0012  0000                 ;ZEUGEN UND DIESE ZUM ÜBEN DES MORSECODES SENDEN. DIE MORSE-
0013  0000                 ;ZEICHEN IN DER TABELLE HABEN FOGENDES FORMAT: VON LINKS NACH
0014  0000                 ;RECHTS IST DIE ERSTE EINS DAS STARBIT, UND AB DANN BEDEUTET
0015  0000                 ;JEDE EINS EINEN STRICH, UND JEDE NULL EINEN PUNKT.
0016  0000                 ;
0017  0000                 SPEED=$F0
0018  0000                 COUNT=$F1
0019  0000                 CHAR=$F2
0020  0000                         *=$300
0021  0300: C9 20          MORSE   CMP #$20        ;FALLS SPACE, SPRINGE ZUR SPACEROUTINE
0022  0302: F0 67                  BEQ SPACE
0023  0304: C9 2C                  CMP #$2C        ;GÜLTIGER ASCII CODE?
0024  0306: 90 4E                  BCC EXIT        ;   FALLS KLEINER ALS 2CH: RETURN
0025  0308: C9 5B                  CMP #$5B        ;GÜLTIGER ASCII CODE?
0026  030A: B0 4A                  BCS EXIT        ;   FALLS GRÖSSER ALS SAH: RETURN
0027  030C: AA                     TAX             ;SCHIEBE CODE INS X-REGISTER
0028  030D: BD 45 03               LDA TABLE-$2C,X ;HOLE MORSEZEICHEN AUS TABELLE
0029  0310: A0 08                  LDY #$8         ;ZAHL DER DURCHZUSCHIEBENDEN BITS
0030  0312: 84 F1                  STY COUNT       ;IN ZÄHLREGISTER COUNT
0031  0314: 0A             STARTB  ASL A
0032  0315: C6 F1                  DEC COUNT
0033  0317: 90 FB                  BCC STARTB      ;SCHIEBE A BIS STARTBIT GEFUNDEN
0034  0319: 85 F2                  STA CHAR
0035  031B: A5 F2          NEXT    LDA CHAR
0036  031D: 0A                     ASL A           ;AUSGABE MORSECODE (1=STRICH, 0=PUNKT)
0037  031E: 85 F2                  STA CHAR
0038  0320: A0 01                  LDY #$1         ;PUNKT = 1 VERZÖGERUNGSEINHEIT
0039  0322: 90 02                  BCC SEND        ;FALLS CARRY CLEAR: SENDE PUNKT
0040  0324: A0 03                  LDY #$3         ;ANDERNFALLS: SENDE STRICH
0041                       ;DIESER TEIL SENDET AM AUSGANG FÜR (Y REGISTER) VERZÖGERUNGS-
0042                       ;EINHEITEN EINE 1, DANN FÜR EINE VERZÖGERUNGSEINHEIT EINE 0.
0043  0326: A9 C0          SEND    LDA #$C0
0044  0328: 8D 0B A0               STA $A00B       ;SETZE ZEITGEBER AUF FREILAUFMODUS
0045  032B: A9 00                  LDA #$0         ;DIESER WERT,
0046  032D: 8D 06 A0               STA $A006
0047  0330: A9 04                  LDA #$04        ;UND DIESER WERT, BESTIMMEN DIE TONHÖHE
0048  0332: 8D 07 A0               STA $A007       ;   DES AUSGANGSSIGNALS (CA 1000 HZ)
0049  0335: 8D 05 A0               STA $A005       ;TON EINSCHALTEN
0050  0338: A9 01                  LDA #$1         ;SCHALTE BIT PBO AM AUSGANG EIN
0051  033A: 8D 00 A0               STA $A000
0052  033D: 20 57 03               JSR DELAY       ;SPRUNG ZUR VERZÖGERUNGSSCHLEIFE
0053  0340: A9 00                  LDA #$0
0054  0342: 8D 0B A0               STA $A00B       ;SCHALTE TON AUS
0055  0345: 8D 00 A0               STA $A000       ;SCHALTE BIT PBO AM AUSGANG AUS
0056  0348: A0 01                  LDY #$01        ;1 VERZÖGERUNGSEINHEIT
0057  034A: 20 57 03               JSR DELAY       ;   FÜR SPACE ZWISCHEN DEN ELEMENTEN
0058  034D: C6 F1                  DEC COUNT       ;DEKREMENTIERE ZÄHLER.PRÜFE,OB 8 BIT FERTIG
0059  034F: D0 CA                  BNE NEXT        ;   FALLS NEIN: NÄCHSTES ELEMENT
0060  0351: A0 02          FINISH  LDY #$2         ;VERZÖGERUNG UM WEITERE 2 EINHEITEN
0061  0353: 20 57 03               JSR DELAY       ;   ALS TRENNUNG ZWISCHEN ZEICHEN
0062  0356: 60             EXIT    RTS
0063                       ;VERZÖGERUNG UM (Y REGISTER) X SPEED X 0.005 SEC
0064  0357: 98             DELAY   TYA
0065  0358: 0A                     ASL A
0066  0359: 0A                     ASL A
0067  035A: A8                     TAY
0068  035B: A5 F0          D3      LDA SPEED
0069  035D: A2 FA          D2      LDX #$FA
0070  035F: CA             D1      DEX
0071  0360: D0 FD                  BNE D1
0072  0362: 38                     SEC
0073  0363: E9 01                  SBC #$1
0074  0365: D0 F6                  BNE D2
0075  0367: 88                     DEY
0076  0368: D0 F1                  BNE D3
0077  036A: 60                     RTS             ;RÜCKSPRUNG AUS VERZÖGERUNGSROUTINE
0078  036B: A0 07          SPACE   LDY #$7         ;VERZÖGERUNG UM 7 EINHEITEN
0079  036D: 20 57 03               JSR DELAY       ;   (ABSTAND ZWISCHEN WORTEN)
0080  0370: 60                     RTS             ;RÜCKSPRUNG AUS MORSEPROGRAMM
```

Bild 4.31: Das Morseprogramm (Vollständiger Ausdruck im Anhang C)

```
0081   0371: 73          TABLE   .BYTE $73,$31,$6A,$32,$3F,$2F
0081   0372: 31
0081   0373: 6A
0081   0374: 32
0081   0375: 3F
0081   0376: 2F
0082   0377: 27                  .BYTE $27,$23,$21,$20,$30,$38
0082   0378: 23
0082   0379: 21
0082   037A: 20
0082   037B: 30
0082   037C: 38
0083   037D: 3C                  .BYTE $3C,$3E,$01,$01,$01,$01
0083   037E: 3E
0083   037F: 01
0083   0380: 01
0083   0381: 01
0083   0382: 01
0084   0383: 01                  .BYTE $01,$4C,$01,$05,$18,$1A
0084   0384: 4C
0084   0385: 01
0084   0386: 05
0084   0387: 18
0084   0388: 1A
0085   0389: 0C                  .BYTE $0C,$02,$12,$0E,$10,$04
0085   038A: 02
0085   038B: 12
0085   038C: 0E
0085   038D: 10
0085   038E: 04
0086   038F: 17                  .BYTE $17,$00,$14,$07,$06,$0F
0086   0390: 00
0086   0391: 14
0086   0392: 07
0086   0393: 06
0086   0394: 0F
0087   0395: 16                  .BYTE $16,$10,$0A,$08,$03,$09
0087   0396: 10
0087   0397: 0A
0087   0398: 08
0087   0399: 03
0087   039A: 09
0088   039B: 11                  .BYTE $11,$08,$19,$18,$1C
0088   039C: 08
0088   039D: 19
0088   039E: 18
0088   039F: 1C


SYMBOL TABELLE:
 SPEED        00F0       COUNT      00F1       CHAR       00F2
 MORSE        0300       STARTB     0314       NEXT       0318
 SEND         0326       FINISH     0351       EXIT       0356
 DELAY        0357       03         0358       02         035D
 01           035F       SPACE      0368       TABLE      0371
```

Bild 4.31: Das Morseprogramm (Fortsetzung)

Der Ton muß am Ausgang für eine Zeitdauer, die wir hier „VERZÖGERUNG“ nennen wollen, anliegen. Man kann diese Zeitverzögerung durch Sorftware- und Hardwaretechniken erreichen. In unserem Programm werden wir eine Softwareschleife benutzen. Schließlich muß der Ton abgeschaltet werden, sobald die vorgegebene Zeitverzögerung erreicht ist. Dazu muß nur Bit 7 des ACR auf 0 gesetzt werden.

Der Leser sollte das Flußdiagramm in Bild 4.28 studieren und sicher sein, daß er die zum Betrieb des Zeitgebers nötige Schrittfolge verstanden hat. Die tatsächliche Implementierung des Programmes werden wir zusammen mit dem Hauptprogramm vorstellen.

Das Morseprogramm

Wir werden nun dem Flußdiagramm in Bild 4.19 folgen und das entsprechende Programm entwickeln. Bei diesem Programm werden wir einige Spezialtechniken anwenden:

Indizierte Adressierung werden wir anwenden, um für ein vorgegebenes ACSII-Zeichen den Binärkode des Morsezeichens aus der Tabelle zu holen.

Den Hardwarezeitgeber benutzen wir zur Erzeugung eines Tones vorgegebener Frequenz. Um die Dauer des Tones zu regulieren, verwenden wir eine Softwaretechnik.

Verschachtelte Schleifen ermöglichen eine Multiplikation in den Verzögerungsschleifen.

Untersuchen wir nun das Programm. Wir nehmen an, daß der Akkumulator schon mit dem Zeichen, dessen Morse-Kode ausgesendet werden soll, geladen ist. Um eine höhere Flexibilität zu erreichen, kann die Sendegeschwindigkeit reguliert werden. Sie wird in Einheiten von 5 Millisekunden (= 0,005 sec) ausgedrückt. Die Variable „GESCHWINDIGKEIT" im Speicher „OOFO" (siehe Speicherbelegung Bild 4.18) muß vor dem ersten Aufruf dieses Programmes definiert werden. Hat die Variable „GESCHWINDIGKEIT" beispielsweise den Wert 20, so wird die Länge eines Punktes („·") 20 x 0,005 sec = 0,1 sec sein. Wir legen das Programm auf die Seite 3, die Startadresse ist „0300" hexadezimal.

Das Programm beginnt folgendermaßen:

```
SPEED    = $00F0    (Geschwindigkeit)
COUNT    = $00F1    (Zähler)
CHAR     = $00F2    (Zeichen)
*        = $0300
```

Diese ersten vier Zeilen sind Anweisungen an den Assembler. Die ersten drei Anweisungen weisen den Variablen SPEED (= GESCHWINDIGKEIT), COUNT (= ZÄHLER) und CHAR (= ZEICHEN) jeweils die Speicherplätze 00F0, 00F1 und 00F2 zu. Die vierte Anweisung legt den Wert des Pseudo-Adreßzählers auf hexadezimal 0300 fest, d. h. sie legt fest, daß die erste auszuführende Programminstruktion bei der Adresse 0300 stehen soll.

Als erstes müssen wir prüfen, ob das im Akkumulator stehende Zeichen zulässig ist. Wir machen das folgendermaßen:

```
MORSE    CMP #$20    IST ES EIN LEERZEICHEN?
         BEQ SPACE
         CMP #$2C    FEHLER FALLS KLEINER 2C
         BCC EXIT
```

```
        CMP  #$5B       FEHLER FALLS GRÖSSER 5B
        BCS  EXIT
```

Die beiden ersten Zeilen prüfen, ob das Zeichen im Akkumulator ein Leerzeichen („space) ist (hexadezimal 20). Falls ja, wird eine reine Verzögerung von 7 Perioden eingefügt.

Die nächsten vier Instruktionen kontrollieren, ob der ASCII-Code zwischen einschließlich „2C" und „5A" liegt. Das ist der Bereich gültiger ASCII-Zeichen für die Morseaussendung. Falls ein ungültiges Zeichen entdeckt wird, liegt der Fehler vor und das Programm springt zur Adresse „EXIT". Um das Programm einfach und für den Lernenden überschaubar zu halten, wird im Programmteil EXIT nichts zur Anzeige des Fehlers unternommen. Wir empfehlen dem Leser jedoch sehr, als Übungsaufgabe bei der Adresse EXIT Befehle einzufügen, die anzeigen, daß ein ungültiges Zeichen im Akkumulator gefunden wurde. In dem vorliegenden Programm werden wir statt des ungültigen Zeichens einfach nichts aussenden.

Wurde im Akkumulator ein gültiges Zeichen vorgefunden, so muß dieses in den entsprechenden Binärcode umgesetzt werden, der dann zur Aussendung der Tonfolge benötigt wird. Der binäre Morsecode, der jedem gültigen ASCII-Zeichen zugeordnet ist, ist am Ende des Programms von der Speicheradresse 0371 bis 039F gespeichert. Für das ASCII-Zeichen 2C wollen wir das erste Byte der Tabelle haben, für das darauffolgende ASCII-Zeichen das nächste Byte, usw. Dies ist ein typischer Fall, bei dem wir die *indizierte Adressierung* verwenden können. Es ergibt sich jedoch ein kleines Zusatzproblem: die ASCII-Zeichen beginnen erst mit 2C und

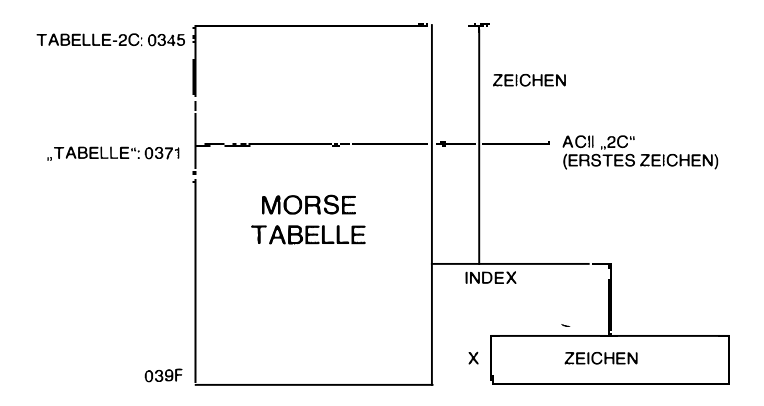


Bild 4.32: Verwendung indizierter Adressierung zur Dekodierung des binären Morse-Kodes

nicht mit 00 oder 01. Dafür gibt es eine recht einfache Lösung, die wir hier vorstellen:

```
TAX
LDA  TABELLE-$2C,X
```

Das ASCII-Zeichen wird in das Indexregister X geschoben, so daß es als Zeiger dienen kann. Um zu berücksichtigen, daß die ASCII-Zeichen erst bei 2C beginnen, wird als Anfangsadresse der Morsetabelle nicht die tatsächliche Anfangsadresse 0371, sondern diese Adresse minus 2C eingesetzt (hexadezimal). Der binäre Morsecode kann dann mit einem einzigen indizierten Speicherzugriff in den Akkumulator geladen werden (siehe Bild 4.32).

Unser binärer Morse-Kode befindet sich jetzt im Akkumulator. Erinnern wir uns, daß dem eigentlichen Morsezeichen eine führende 1 als Startbit vorausgeht. Dem Startbit folgen die Nullen und Einsen, die die Punkte und Striche darstellen. Jedes der links vom Startbit stehenden und nicht genutzten Bits ist auf 0 gesetzt. Der Inhalt des Akkumulators wird daher solange nach links geschoben, bis das Startbit gefunden wird. Die dann noch verbleibenden „echten" Bits entsprechen den Punkten und Strichen und werden zur Tonerzeugung verwendet. Hier das Programm:

```
        LDY  #$08     ANZAHL DER DURCH A
                      DURCHZUSCHIEBENDEN
                      BITS
        STY  COUNT    SPEICHERE IN ,ZÄHLER'
STARB   ASL  A
        DEC  COUNT    DEKREMENTIERE
                      ,ZÄHLER'
        BCC  STARTB   SCHIEBE A NACH LINKS BIS
                      STARTBIT GEFUNDEN
        STA  CHAR
NEXT    LDA  CHAR
        ASL  A        ERSTES MORSEBIT NACH C
        STA  CHAR     REST NACH ,ZEICHEN'
        LDY  #$01     PUNKT = 1 PERIODE
        BCC  SEND     FALLS C = 0: PUNKT
        LDY  #$03     SONST STRICH
                      (3 PERIODEN)
```

Normalerweise würde man das Y-Register als Zähler benutzen, um das wiederholte nach links Durchschieben des Akkumulatorinhaltes nach dem achten Bit zu stoppen. Die Routine SEND, die die Töne erzeugt, lädt jedoch das Y-Register mit der Dauer des auszusendenden Tones. Wir

können daher zum Durchschieben der Bits das Indexregister Y nicht verwenden. Die nächste Idee, in solch einer Situation wäre die Verwendung des jetzt nicht mehr gebrauchten X-Registers. Leider ist unser Programm aber so gestaltet, daß die Routine DELAY auf das X-Register zugreift. Da also keines der beiden Indexregister als Zähler zur Verfügung steht, werden wir einen Speicherplatz als Zähler vereinbaren müssen. Es ist dies der Speicherplatz ‚COUNT' bzw. „ZÄHLER". Es ist sehr wichtig zu bemerken, daß wir beim Programmieren diesen Teil des Programmes wahrscheinlich vor der Programmierung der Routinen SEND und DELAY erstellt hätten. Wir hätten daher sicherlich eines der Indexregister X oder Y zum Abzählen der aus dem Akkumulator bereits herausgeschobenen Bits verwendet. Erst später hätten wir festgestellt, daß dasselbe Register in der Routine SEND bzw. DELAY verwendet werden muß. An diesem Punkt kommt eine gute Programmierdisziplin voll zum Tragen. Wenn man feststellt, daß andere Routinen Zugriff auf die Register X und Y benötigen, muß man im Assemblerprogramm zurückgehen und dieses derart abändern, daß man statt des Indexregisters den Speicher ‚ZÄHLER' verwendet. Leider ist es ein klassischer Programmierfehler, dies zu vergessen. In diesem Fall wird die andere Routine den Inhalt des jeweiligen Indexregisters X oder Y durch seinen Zugriff zerstören. Dies führt im allgemeinen zu äußerst üblen Programmierfehlern. Im Einklang mit einer guten Programmierdisziplin empfiehlt es sich daher, *am Beginn jeder Routine ganz explizit zu kommentieren, welche Register von der Routine verändert oder zerstört werden*. Vor dem Schreiben einer neuen Routine sollten die Vereinbarungen über Kommunikation und Datenaustausch zwischen den Unterprogrammen oder Programmteilen vollständig abgeklärt sein.

Die im Akkumulator links stehenden Nullen werden unterdrückt und der Akkumulatorinhalt wird so lange nach links durchgeschoben, bis das Startbit gefunden ist. Ist das Startbit gefunden, stellt jedes weitere Bit, das aus dem Akkumulator nach links herausgeschoben wird, entweder einen Punkt oder einen Strich dar, je nachdem ob es sich um eine 0 oder eine 1 handelt. Sobald das aus dem Akkumulator herausgeschobene Bit identifiziert ist, führen wir einen Sprung zum Programmteil SEND (= AUSSENDUNG) durch, wo ein entsprechend langer Ton erzeugt wird. Da der Akkumulatorinhalt durch die dabei ablaufenden Vorgänge geändert wird, müssen wir ihn vor dem Sprung zur Routine SEND retten. Das erledigt die nächste Instruktion STA CHAR. Wenn wir den Akkumulatorinhalt so in den Speicher CHAR (= ZEICHEN) gerettet haben, laden wir das Indexregister Y mit einem Wert, die der Länge des auszusendenden Tones entspricht. Der Wert wird von dem gerade aus dem Akkumulator herausgeschobenen Bit bestimmt: war es ein Punkt, so wird Y mit einer „1" geladen, war es ein Strich, dann mit einer „3". Die beiden nachfolgenden Instruktionen STA CHAR und LDA CHAR scheinen unsinnig, wir brauchen sie aber, um später an der Stelle „NEXT" mit einer LDA CHAR-Instruktion wieder in das Programm springen zu können.

Die SEND-Routine

Die SEND-Routine verwendet zur Erzeugung eines Tons gegebener Frequenz den Zeitgeber 1 des 6522. Bild 4.33 zeigt das Registerverzeichnis dieses Zeitgebers. Der Zeitgeber muß im Freilaufmodus betrieben werden. Das macht man folgendermaßen:

	ADRESSE	SCHREIBEN	LESEN
ZEITGEBER 1	--04	T1L-L	T1C-L/ + lösche T1 Interrupt-Flag
	--05	T1L-H-->T1C-H T1L-L-->T1C-L + lösche T1 Interrupt-Flag	T1C-H
	--06	T1L-L	T1L-L
	--07	T1L-H + lösche T1 Interrupt-Flag	T1L-H

Bild 4.33: Speicherbelegung des Zeitgebers 1 des 6522

```
SEND        LDA   #$C0
            STA   $A00B
```

Das Byte C0 wird in den Speicher A00B geschrieben. A00B ist das ACR oder Hilfssteuerregister. Hierdurch werden die Bits 6 und 7 entsprechend gesetzt (für Details siehe Bild 4.29 und 4.30). Dann wird der Wert 0400 hexadezimal in die Speicherstellen A006 und A007 geschrieben:

```
            LDA   #$00
            STA   $A006
            LDA   #$04
            STA   $A007
```

Das sind die Adressen des niederwertigen und des höherwertigen Bytes des Zwischenspeichers T1L. Durch diese Befehlsfolge wird die Frequenz des erzeugten Tones festgelegt. 0400 hexadezimal ist binär: 00000100 00000000 oder dezimal: 1024. Die halbe Schwingungsdauer ist ca. N+2 mal die Dauer eines Systemtaktes, also etwa das 1026fache. Die gesamte Schwingungsdauer T ist daher:

$$T = 2052\ \mu sec$$

Und die Frequenz ist der Kehrwert davon:

$$N = 1/T \approx 500\,Hz.$$

Nun müssen wir den Ton starten und nach einer bestimmten Zeit wieder ausschalten. Der Ton wird eingeschaltet mit:

```
            STA   $A005
```

Dieser Befehl überträgt den Inhalt des Zwischenspeichers in das Zählregister und startet die Tonerzeugung. Wir hatten angedeutet, daß das Programm auch manuel den Ausgang PB0 einschalten soll, damit ein externes Gerät wie etwa ein Sender gleichzeitig mit dem Beginn der Tonerzeugung eingeschaltet werden kann. Das machen wir so:

```
LDA  #$01
STA  $A000
```

Dabei wird angenommen, daß PB0 schon vor dem ersten Aufruf dieses Programmes als Ausgabekanal festgelegt worden war.

Die Dauer des Tones wird durch das Unterprogramm DELAY festgelegt: JSR DELAY. Wir werden dieses Unterprogramm weiter unten untersuchen. Sobald die vorgesehene Zeit um ist, muß der Ton ausgeschaltet werden:

```
LDA  #$00
STA  $A00B     SCHALTET TON AB
STA  $A000     SCHALTET PB0 AB
```

Schließlich müssen wir den Tongenerator zwischen zwei Tönen für eine Periode abgeschaltet lassen:

```
LDY  #$01
JSR  DELAY     VERZÖGERUNG VON
               1 PERIODE
```

Nun müssen wir unseren Bitzähler, den Speicher ZÄHLER (COUNT), dekrementieren, um zu prüfen, ob noch weitere Bits durch den Akkumulator geschoben werden müssen:

```
DEC  COUNT     8 BIT DURCHGESCHOBEN?
BNE  NEXT      FALLS NICHT:
               NÄCHSTES BIT
```

Wenn das Zeichen vollständig ausgesendet ist, müssen noch zwei weitere Verzögerungsperioden angefügt werden, um das Zeichen vom nachfolgenden Zeichen zu trennen:

```
FINISH    LDY  #$02
          JSR  DELAY
EXIT      RTS
```

Das Unterprogramm DELAY

Dieses Verzögerungs-Unterprogramm erzeugt eine Verzögerungszeit t von:

$$t = (\text{Inhalt des Y-Registers}) \times (\text{SPEED}) \times 0{,}005 \text{ sec}$$

Die Dauer der gewünschten Verzögerung berechnet sich daher aus der Multiplikation dreier Zahlen. Wir werden hier eine spezielle Technik der

verschachtelten Schleifen anwenden, um die Ausführung einer klassischen Multiplikation zu vermeiden. Hier die Routine:

```
DELAY   TYA
        ASL  A        A = A x 2
        ASL  A        A = A x 2
        TAY           Y = Y x 4
D3      LDA  SPEED
D2      LDX  #$FA
D1      DEX
        BNE  D1
        SEC
        SBC  #$01
        BNE  D2
        DEY
        BNE  D3
        RTS
```

Das zugehörige Flußdiagramm zeigt Bild 4.34.

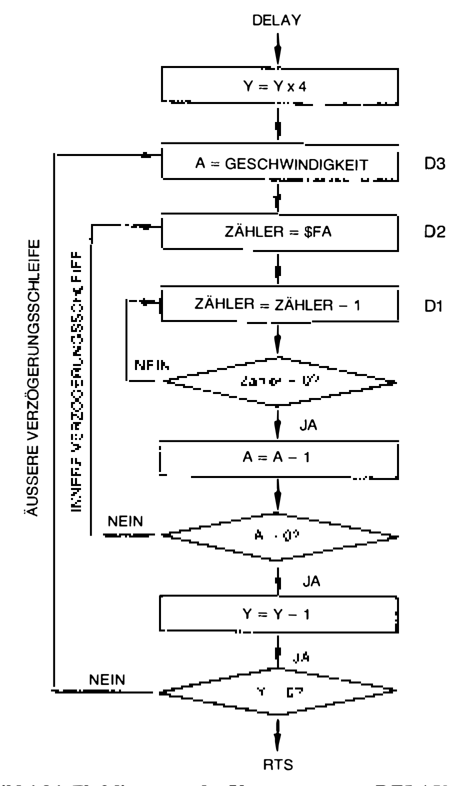


Bild 4.34: Flußdiagramm des Unterprogramms DELAY

Die ersten vier Zeilen der DELAY-Routine vervierfachen den Inhalt des Y-Registers:

```
DELAY       TYA
            ASL  A          A = A * 2
            ASL  A          A = A * 2
            TAY             Y = Y * 4
```

Eine Vervierfachung entspricht einer zweimal hintereinander ausgeführten Verdoppelung. Eine Verdoppelung einer binär dargestellten Zahl wiederum läßt sich durch simples nach links schieben um 1 Bit erreichen (Division durch 2 entspricht dem Verschieben um 1 Bit nach rechts, auftretende Reste gehen dabei aber verloren). Da Schiebebefehle nur für den Akkumulator existieren, muß der Inhalt des Y-Registers erst mit TYA in den Akkumulator gebracht werden. Die doppelte Anwendung des ASL A-Befehles vervierfacht den Akkumulatorinhalt. Ein Übertrag kann nicht auftreten, da die größte auftretende Zahl 7 ist (SPACE-Routine) und 7 x 4 kleiner als 255 ist. Der Befehl TAY transportiert anschließend den Inhalt des Akkumulators wieder in das Y-Register zurück.

Wir wollen nun daran gehen, die genaue Dauer dieser Softwareverzögerung zu berechnen. Die Zahlen in Klammern geben die Anzahl der von dem Befehl benötigten Taktzyklen an.

```
(2)               TYA
(2)               ASL  A
(2)               ASL  A
(2)               TAY
(3)      D3       LDA  SPEED
(2)      D2       LDX  #$FA       FA HEX = 250 dezimal
(2)      D1       DEX
(3/2)             BNE  D1
```

Die Schleife D1 hat eine Länge von 250 x (2+3) − 1 Taktzyklen = 1249 µsec. Der letzte Befehl BNE benötigt 3 Takte, wenn der Sprung ausgeführt wird, andernfalls nur 2. Beim letzten Schleifendurchlauf wird daher ein Takt weniger gebraucht.

Die beiden nächsten Instruktionen sind:

```
(2)               SEC
(2)               SBC  #$01
```

Jeder dieser beiden Befehle benötigt 2µsec. Insgesamt addiert sich zur bisherigen Verzögerung also ein Betrag von weiteren 4 µsec. Diese beiden Befehle subtrahieren die Zahl 1 vom Akkumulator. Man muß diesen Weg beschreiten, da sowohl das X- als auch das Y-Register in diesem Programm schon als Zähler belegt sind. Daher muß der Akkumulator als drittes Zählregister herangezogen werden. Leider gibt es keinen Dekrementierbefehl, der direkt auf den Akkumulator wirkt. Deswegen muß ei-

ne formale Subtraktion durchgeführt werden. Der Leser wird sich erinnern, daß vor einer Subtraktion das Übertragsflag (carry) gesetzt werden muß. Dafür steht der SEC-Befehl vor SBC. Es folgt der nächste Befehl:

```
(3/2)          BNE  D2
```

Dies ist eine zweite Verzögerungsschleife. Wird der Sprung ausgeführt, so benötigt dies 3 µsec, wird er nicht ausgeführt, dann nur 2 µsec. Gerechnet ab Beginn der inneren Schleife D2 sind das bisher insgesamt $2+1249+2+2+3 = 1258$ µsec.

Bei jedem Durchlaufen der Schleife wird diese Verzögerung erreicht. Die Schleife wird so oft durchlaufen, wie die Variable SPEED (GESCHWINDIGKEIT) angibt. Das ergibt dann, so wie wir es wollten, eine Verzögerungszeit von SPEED x 1258 µsec. Eine dritte Schleife wird so oft durchlaufen, wie der Inhalt des Y-Registers angibt:

```
               DEY
               BNE  DELAY
               RTS
```

Ganz zu Anfang war das Y-Register vervierfacht worden, so daß wir nach dieser letzten Schleife die gewünschte Gesamtverzögerung von 1258 µsec x SPEED x 4 x Y oder von näherungsweise 0,005 sec x SPEED x Y erhalten. Nach dieser Verzögerung erfolgt der Rücksprung (RTS).

Anwendung des Programms. Um das Programm zu benutzen, sollte man zu Beginn eine recht langsame Sendegeschwindigkeit wählen, falls man mit dem Morse-Kode nicht sehr gut vertraut ist. Erzeugen Sie auch immer nur ein Zeichen auf ein Mal. Wenn Sie sich dann überzeugt haben, daß das Programm richtig arbeitet, sollten Sie ein kurzes Unterprogramm schreiben, das Zeichen an das Morseprogramm übergibt. Sie können sich dann überzeugen, daß das Programm auch für ganze Zeichenketten richtig arbeitet.

Übungsaufgabe 4.2: *Schreiben Sie ein Unterprogramm, das eine Zeichenkette von N Zeichen an das Morseprogramm übergibt. Die Zeichenkette soll in einer Tabelle mit der Startadresse TABELLE abgelegt sein.*

Übungsaufgabe 4.3: *Lesen Sie Zeichen über die Tastatur ein und erzeugen Sie die entsprechenden Morsezeichen.*

24-Stunden-Uhr

Wir werden ein Uhrzeitprogramm entwickeln, das die aktuelle Tageszeit in Stunden, Minuten und Sekunden in drei vereinbarten Speichern angibt. Bei Bedarf kann das Programm problemlos so erweitert werden, daß es auch Bruchteile von Sekunden abspeichert oder in anderen Zeiteinheiten zählt. Bild 4.35 zeigt, wie das Programm den Speicherbereich belegt. Für die Variablen wurden wie gewöhnlich Speicherplätze in der Seite 0

reserviert. Die Stunden, Minuten und Sekunden werden in die Speicher 00F4, 00F5 und 00F6 (hexadezimal) gelegt. Es wird ein weiterer Speicher benötigt: in 00F7 steht die Variable COUNT (ZÄHLER).

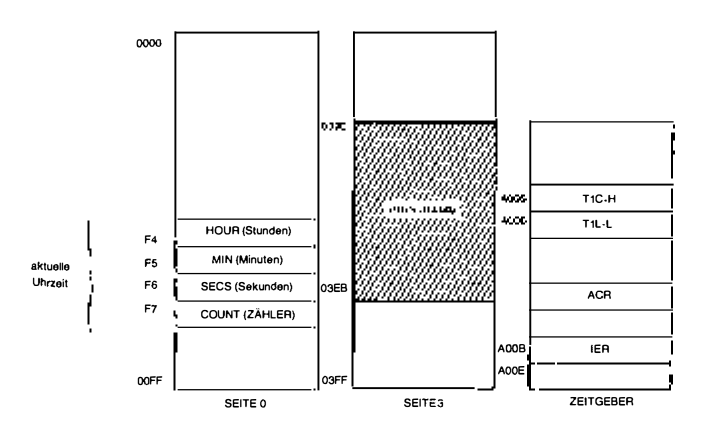


Bild 4.35: 24-Stunden-Uhr, Speicherbelegung

Um die Uhr zu starten, geben wir das Progamm ein und speichern dann die aktuelle Uhrzeit plus eine Minute in die Speicher SECS (Sekunden), MIN (Minuten) und HOURS (Stunden).

Dann speichert man A7 in A67E und 03 in A67F (nur für SYM). Dies ist der Interruptvektor; wir kommen später darauf zurück. Dann geben wir „GO 0390" ein und drücken, sobald die Uhrzeit, die wir vorher eingegeben hatten, erreicht ist, auf „CR".

Von diesem Zeitpunkt an hat die Uhr die genaue Zeit in Sekunden, Minuten und Stunden in den Speichern SECS, MIN und HOURS.

In der Variablen COUNT wird in Einheiten von einer zwanzigstel Sekunde gezählt. Sie wird mit dem Wert 20 initialisiert und dann jede zwanzigstel Sekunde dekrementiert. Das Dekrementiersignal ist ein Hardwareinterrupt, der von einem der 6522-Zeitgeber ausgelöst wird. Bild 4.36 zeigt das Flußdiagramm für die 24-Stunden-Uhr. Der erste Teil dient der Initialisierung. Das Zählregister des Zeitgebers wird dabei mit einem Startwert geladen, so daß nach 50 Millisekunden (1/20 sec) ein Interrupt ausgelöst wird. Dann wird die Variable COUNT mit dem Wert 20 (dezimal) initialisiert und der Zeitgeber gestartet.

Wenn der Zeitgeber ausgezählt hat, ist eine zwanzigstel Sekunde vorbei und der Zeitgeber setzt einen Interrupt. Nach diesem Interrupt wird der Mikroprozessor sein Statusregister und den Akkumulator retten, den Zähler des Zeitgebers wieder mit dem für 50 msec nötigen Wert laden

und sofort wieder starten. Wenn eine zwanzigstel Sekunde verstrichen ist, wird die Variable COUNT dekrementiert. Der neue Wert von COUNT wird auf Null getestet. Ist er nicht „0“, springt der Prozessor aus dieser Interrupt-Routine zurück ins Hauptprogramm. Hat COUNT den Wert „0“ erreicht, wird er wieder auf „20“ zurückgesetzt und der Speicher SECS wird inkrementiert.

Wenn SECS inkrementiert ist, wird auf den Wert „60“ getestet. Ist dieser Wert erreicht, wird SECS auf „0“ zurückgesetzt und MIN (der Minutenzähler) wird inkrementiert. Genauso wird dann MIN auf den Wert „60“ getestet und die Variable HOURS wird inkrementiert. Die Variable HOURS wird aber schon beim Erreichen von 24 auf „0“ zurückgesetzt. Dann erfolgt der Rücksprung. Das Programm bleibt bis zum Auftreten des nächsten Interrupts unbenutzt. Der Benutzer muß zur Anzeige der aktuellen Uhrzeit lediglich die Inhalte der Speicher F4, F5 und F6 lesen. Man könnte auch eine kurze Routine schreiben, die diese Inhalte automatisch anzeigt.

Wir zeigen das Programm in Bild 4.37 und es dürfte sich von selbst erklären. Der erste Abschnitt des Programms ist das Startprogramm INIT, welches die Variable COUNT mit dem Startwert 20 (dezimal) lädt (= 14 hexadezimal) und außerdem den Zeitgeber mit dem für eine Verzögerung von 50 msec nötigen Zählerinhalt lädt. Die zugehörige Speicherbelegung finden Sie in Bild 4.35. Es wird der Zeitgeber 1 des 6522 verwendet. Die Bedeutung der einzelnen Bits des ACR zeigen Bild 4.25 und Bild 4.29. Man kann den Zeitgeber sowohl im Einzelbetrieb als auch freilaufend benutzen. Beim Einzelbetrieb wird jedesmal, wenn der Zähler des Zeitgebers bei 0 angelangt ist, ein einzelner Interrupt (und eventuell ein Signal auf PB7) erzeugt. Im Freilaufmodus wird der Zähler mit dem Inhalt des Zwischenspeichers automatisch neu geladen und es werden fortlaufend Interrupts (und eventuell Ausgangssignale an PB7) erzeugt. Da PB7 in diesem Beispiel nicht benutzt wird, setzen wir Bit 7 des ACR auf Null. Dann muß man sich noch auf Einzelbetrieb oder Freilaufmodus festlegen. Beim Einzelbetrieb muß der Zähler nach jedem Interrupt neu geladen werden. Im freilaufenden Betrieb wird der Zeitgeber sein internes Zählregister mit dem Inhalt des Zwischenspeichers automatisch neu laden. Jedoch muß man dann das Interruptflag explizit löschen, indem man entweder in T1CH schreibt, oder das Flag direkt zurücksetzt. Im Hinblick auf den Programmieraufwand sind beide Verfahren vergleichbar. Der Freilaufmodus wird eine genauere Zeitmessung ergeben, da der Zeitgeber durchläuft und automatisch vom Wert „0“ auf den Wert, der 50 msec entspricht, zurückspringt. Da der Freilaufmodus schon im Morseprogramm studiert wurde, wollen wir diesmal den Einzelbetrieb wählen. Als Übung sollte der Leser aber versuchen, das andere Verfahren auch zu verwenden. Der Einzelbetrieb wird definiert, indem das Bit 6 des ACR auf „0“ gesetzt wird. Alle anderen Bits des ACR werden nicht gebraucht und können daher auch auf „0“ gesetzt werden. Die Bits 6 und 7 sind damit auf „0“, wodurch Einzelbetrieb mit gesperrtem PB7 festgelegt ist.

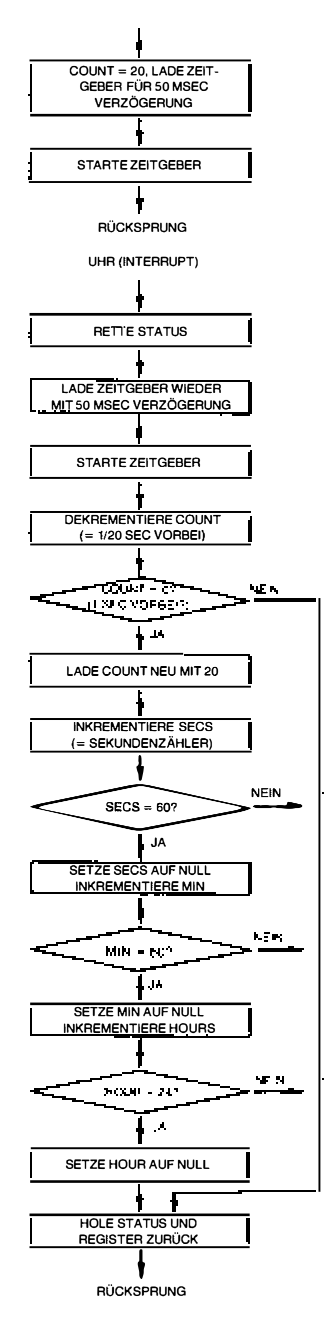


Bild 4.36: 24-Stunden-Uhr, Flußdiagramm

```
ZEILE ADR       CODE        ZEILE

0002  0000                 ;LADEN SIE ZUERST A7 IN SPEICHER A67E, UND 03 IN SPEICHER A67F.
0003  0000                 ;DIES IST EIN UNTERPROGRAMM FÜR EINE ECHTZEITUHR, DIE DIE AKTUELLE
0004  0000                 ;UHRZEIT IN DIE SPEICHER SECS(00F6), MIN (00F5), UND HOUR (00F4)
0005  0000                 ;SPEICHERT (24 STUNDEN UHR). DURCH DEN INTERRUPT BEIM ABLAUFEN DES
0006  0000                 ;ZEITGEBERS WIRD 20 MAL IN DER SEKUNDE EIN SPRUNG ZUM UHRZEITPRO-
0007  0000                 ;GRAMM BEWIRKT. ALS ERSTES MÜSSEN DER INTERVALLZEITGEBER UND DAS
0008  0000                 ;UHRENPROGRAMM INITIALISIERT WERDEN. DIES MACHT DER TEIL 'INIT'.
0009  0000                 ;ZUM STARTEN DES PROGRAMMS MUSS ER ANGESPRUNGEN WERDEN. GEBEN SIE
0010  0000                 ;DIE GENAUE UHRZEIT, BEI DER DIE UHR GESTARTET WERDEN SOLL, IN
0011  0000                 ;DIE SPEICHER SEC, MIN, HOUR EIN, UND STARTEN SIE DAS PROGRAMM
0012  0000                 ;ZU DIESEM ZEITPUNKT MIT 'GO 0390 CR'. MEHR IST NICHT ZU TUN.
0013  0000                 COUNT=$00F7              ;ZÄHLER FÜR ZWANZIGSTELSEKUNDEN
0014  0000                 SECS=$00F6               ;AKTUELLE UHRZEIT (SEKUNDEN)
0015  0000                 MIN=$00F5                ;                 (MINUTEN)
0016  0000                 HOUR=$00F4               ;                 (STUNDEN)
0017  0000                 ACR=$A00B                ;HILFSSTEUERREGISTER FÜR ZEITGEBERMODUS
0018  0000                 T1LL=$A006               ;ZEITKONSTANTE, NIEDERWERTIGES BYTE
0019  0000                 T1CH=$A005               ;ZEITKONSTANTE, HÖHERWERTIGES BYTE
0020  0000                 *=$0390
0021  0390  A9 14          INIT   LOA #$14          ;SETZE ZÄHLER AUF 20
0022  0392  BS F7                 STA COUNT
0023  0394  BO 0B AO              STA ACR           ;SETZE BI[illegible]CR AUF 0
0024  0397  A9 CO                 LOA #$CD          ;SETZE BIT 6 [illegible]ER AUF 1
0025  0399  BO OE AD              STA $AODE         ;  DAMIT IN[illegible]S ZUGELASSEN SIND
0026  039C  A9 50                 LOA #$50          ;LADE ZEITGEBERREGISTER MIT KONSTANTE
0027  039E  BO 06 AD              STA T1LL          ;  C350 (VERZÖGERUNGSKONSTANTE)
0028  03A1  A9 C3                 LOA #$C3          ;  ENTSPRECHEND 50 MSEC
0029  03A3  BO 05 AD              STA T1CH          ;HIER WIRD DER ZEITGEBER GESTARTET
0030  03A6  60                    RTS               ;RÜCKSPRUNG IN MONITOR
0031  03A7  08             CLOCK  PHP               ;RETTE STATUSREGISTER
0032  03A8  48                    PHA               ;RETTE AKKUMULATOR
0033  03A9  F8                    SED
0034  03AA  A9 50                 LOA #$50          ;LADE ZEITGEBERREGISTER NEU MIT C350
0035  03AC  BO 06 AD              STA T1LL          ;  (VERZÖGERUNGSKONSTANTE FÜR 50 MSEC)
0036  03AF  A9 C3                 LOA #$C3
0037  03B1  BO 05 AD              STA T1CH          ;HIER WIRD DER ZEITGEBER GESTARTET
0038  03B4  C6 F7                 DEC COUNT         ;DEKREMENTIERE ZÄHLER FÜR 20TEL SEC
0039  03B6  DO 31                 BNE EXIT          ;BEI WENIGER ALS 20 ZYKLEN RÜCKSPRUNG
0040  03B8  A9 14                 LOA #$14          ;ANDERNFALLS IST 1 SEC VOLL - ZÄHLER
0041  03BA  BS F7                 STA COUNT         ;  WIEDER NEU AUF 20 SETZEN
0042  03BC  A9 01                 LOA #$01
0043  03BE  18                    CLC
0044  03BF  65 F6                 ADC SECS          ;ADDIERE 1 ZU DEN SEKUNDEN
0045  03C1  BS F6                 STA SECS
0046  03C3  C9 60                 CMP #$60          ;60 SEKUNDEN VOLL?
0047  03C5  DO 22                 BNE EXIT          ;FALLS NEIN, RÜCKSPRUNG
0048  03C7  A9 00                 LOA #$00          ;ANDERNFALLS SETZE SECS AUF 0 ZURÜCK
0049  03C9  BS F6                 STA SECS
0050  03CB  A9 01                 LOA #$01
0051  03CD  18                    CLC
0052  03CE  65 F5                 ADC MIN           ;ADDIERE 1 ZU DEN MINUTEN
0053  03D0  B5 F5                 STA MIN
0054  03D2  C9 60                 CMP #$60          ;60 MINUTEN VOLL?
0055  03D4  DO 13                 BNE EXIT          ;FALLS NEIN, RÜCKSPRUNG
0056  03D6  A9 00                 LOA #$00          ;ANDERNFALLS SETZE MIN AUF 0 ZURÜCK
0057  03D8  BS F5                 STA MIN
0058  03DA  A9 01                 LOA #$01
0059  03DC  18                    CLC
0060  03DD  65 F4                 ADC HOUR          ;ADDIERE 1 ZU DEN STUNDEN
0061  03DF  BS F4                 STA HOUR
0062  03E1  C9 24                 CMP #$24          ;24 STUNDEN VOLL?
0063  03E3  DO 04                 BNE EXIT          ;FALLS NEIN, RÜCKSPRUNG
0064  03E5  A9 00                 LOA #$00          ;ANDERNFALLS SETZE HOUR AUF 0 ZURÜCK
0065  03E7  B5 F4                 STA HOUR
0066  03E9  68             EXIT   PLA               ;HOLE AKKUMULATOR ZURÜCK
0067  03EA  28                    PLP               ;HOLE STATUSREGISTER ZURÜCK
0068  03EB  40                    RTI               ;RÜCKSPRUNG VON INTERRUPT


SYMBOL TABELLE

ACR     A00B    CLOCK   03A7    COUNT   00F7    EXIT    03E9
HOUR    00F4    INIT    0390    MIN     00F5    SECS    00F6
T1CH    A005    T1LL    A006
```

Bild 4.37: das Programm 24-Stunden-Uhr (großformatiges Listing im Anhang C)

Als nächstes muß das Interrupt-Flag-Register geladen werden. Beim Lesen (read, R) ist dieses Register das Interrupt-Flag-Register IFR, beim Schreiben (write, W) ist es das Interrupt-Enable-Register IER. Um einzelne Bits des IER setzen zu können, muß Bit 7 des IER auf „1“ gesetzt werden.Für jede weitere „1“ in den Bitpositionen 0 bis 6 wird in das IER eine „1“ geschrieben und die entsprechende Funktion damit zugelassen (enable). Eine „0“ in einer dieser Bitpositionen wird das entsprechende Bit des IER nicht zurücksetzen, sondern unverändert lassen. Einzelne Bits löscht man, indem man eine „0“ in Bit 7 des IER und Einsen in alle zu löschenden Bitpositionen schreibt. In unserem Beispiel wollen wir lediglich den Interrupt von Zeitgeber 1 zulassen. Wir werden daher in den Speicher, dessen Adresse dem IER entspricht, den Wert „11000000“ binär bzw. „CO“ hexadezimal schreiben (für weitere Einzelheiten siehe Kapitel 2).

Damit der Zeitgeber nach 50 msec einen Interrupt auslöst, müssen wir noch einen geeigneten Wert in das Zählregister laden. Wir schreiben also den Wert C350 hexadezimal (= 50000 dezimal) in das Zählregister. Man beachte, daß in der Routine INIT zuerst das niederwertige Byte des Zwischenspeichers, und dann das höherwertige Byte des Zählers geladen wird. Sobald das höherwertige Byte des Zählers geladen wird, wird das niederwertige Byte automatisch vom Zwischenspeicher in das Zählregister übertragen und gleichzeitig der Zeitgeber gestartet.

Als erstes zeigen wir die Routine INIT:

```
COUNT  = $00F7  1/20 SEKUNDE (ZÄHLER)
SECS   = $00F6  SEKUNDEN
MIN    = $00F5  MINUTEN
HOUR   = $00F4  STUNDEN
ACR    = $A00B  HILFSSTEUERREGISTER
T1LL   = $A006  ZWISCHENSPEICHER, NIEDERWERTIGES
                BYTE
T1CH   = $A005  ZÄHLREGISTER, HOCHWERTIGES BYTE


INIT          LDA  #$14      20 DEZIMAL IN COUNT
              STA  COUNT
              STA  ACR       BITS 6 UND 7 DES ACR
                             AUF „0“
              LDA  #$C0      „11000000“ BINÄR
              STA  $A00E     BIT 6 DES IER AUF „1“
              LDA  #$50      $C350 IN ZÄHLREGISTER
              STA  T1LL      (ERGIBT 50 MSEC)
              LDA  #$C3
              STA  T1CH      STARTE ZEITGEBER
              RTS
```

Die Initialisierung ist nun beendet und das Programm startet an der Adresse CLOCK (UHR). Man beachte, daß innerhalb der Routine CLOCK alle Additionen im Dezimalmodus durchgeführt werden. Aus diesem Grund wird mit dem Befehl SED das Dezimalflag gesetzt. Daher werden die Registerinhalte bei der Ausgabe über ein Display so angezeigt, daß pro LED eine Stelle in der gewohnten dezimalen Form anstatt im hexadezimalen Format gezeigt wird.

Am Ende der Routine INIT steht ein RTS-Befehl und der Prozessor springt in den Monitor zurück. Sofern keine Taste gedrückt ist, passiert nichts, bis der Zeitgeber einen Interrupt auslöst, der anzeigt, daß die vorgegebene Zeit abgelaufen ist. Sobald der Interrupt registriert ist, erfolgt automatisch ein Sprung zur Routine CLOCK. Beim Auftreten eines Interrupts springt der 6502 stets automatisch zu den Adressen FFFE und FFFF, wo der Interruptvektor abgelegt ist. Die Adresse, auf die der Interruptvektor zeigt, muß dann in den internen Programmzähler geladen werden. Beim SYM lädt der Benutzer den gewünschten Interruptvektor in die Speicher A67E und A67F. Der SYM-Monitor, der immer dann läuft, wenn gerade kein Benutzerprogramm abgearbeitet wird, kopiert automatisch die Inhalte der Speicher A600 bis A67F in die Speicher FF80 bis FFFF. Der Inhalt der Speicher A67E und A67F wird daher vom SYM-Monitor automatisch nach FFFE und FFFF übertragen. Sobald ein Interrupt auftritt, erfolgt ein Sprung nach FFFE/FFFF, wo der Prozessor eine 16-Bit Adresse findet, die er in seinen Programmzähler lädt.

Bei jedem Interrupt wird CLOCK als Interruptroutine angesprungen. Die CLOCK-Routine rettet das P-Register (Prozessor Status Register) und A (Akkumulator). Sie braucht die anderen Register nicht zu retten, da diese nicht verwendet werden.

Dann wird das Zählregister des Zeitgebers mit dem Wert C350 hexadezimal (= 50.000 dezimal) neu geladen und der Zeitgeber wieder gestartet. Durch das Laden des Zählregisters wird automatisch der noch bestehende Interrupt gelöscht.

Die Routine überprüft dann nacheinander, ob die Variable COUNT den Wert „00“, die Variable SECS den Wert „60“, die Variable MIN den Wert „60“ und die Variable HOURS den Wert „24“ erreicht hat. Hat die Variable COUNT den Wert „00“ erreicht, so wird sie auf „20“ = hex $14 zurückgesetzt. Hat eine der anderen Variablen den betreffenden Endwert erreicht, so wird sie auf „0“ zurückgesetzt, wie man im Flußdiagramm Bild 4.36 oder im Programmlisting Bild 4.37 sieht.

Zuletzt werden die beiden geretteten Register zurückgeholt (A und P) und es erfolgt der Rücksprung aus der Interruptroutine mit RTI.

Ein Heim-Steuer-Programm

Ein umfassendes Heim-Steuer-Programm besitzt eine interne 24-Stunden-Uhr, testet den Status von Alarmanlagen und führt je nach Tageszeit

oder je nach gemeldetem Alarm verschiedene Schaltungen durch. Wir werden das eben entwickelte Programm „24-Stunden-Uhr" einsetzen und die aktuelle Tageszeit anzeigen. Je nach Tageszeit werden verschiedene Schaltungen durch das Ein- oder Ausschalten von Relais vorgenommen. Das Programm ist in Bild 4.38 aufgelistet. Das Datenrichtungsregister von Tor B wird auf OF hexadezimal gesetzt, um die vier ersten Bits als

```
ZEILE ADR      CODE        ZEILE

0002  0000                ;DIES IST EIN EINFACHES PROGRAMM ZUR HEIM-
0003  0000                ;STEUERUNG, DAS IN EINER ENDLOSSCHLEIFE LÄUFT.
0004  0000                ;BEI JEDEM DURCHLAUF ZEIGT ES DIE AKTUELLE
0005  0000                ;ZEIT AN UND SPRINGT IN EINIGE UNTERPROGRAMME,
0006  0000                ;DIE ANGESCHLOSSENE GERÄTE STEUERN, Z.B.:
0007  0000                ;1) EIN UNTERPROGRAMM KANN DIE UHRZEIT TESTEN
0008  0000                ;   UND ZU EINER BESTIMMTEN ZEIT EINE LAMPE
0009  0000                ;   EIN - ODER AUSSCHALTEN.
0010  0000                ;2) EIN ANDERES UNTERPROGRAMM KÖNNTE DEN STATUS
0011  0000                ;   EINER ALARMANLAGE PRÜFEN UND BEIM AUFTRETEN
0012  0000                ;   EINES ALARMS GEEIGNET REAGIEREN.
0013  0000                DDRB=$AC02
0014  0000                IORB=$AC00
0015  0000                HOUR=$00F4
0016  0000                MIN=$00F5
0017  0000                OUTBYT=$82FA
0018  0000                SCAND=$8906
0019  0000                       *=$0200
0020  0200  D8            CONTRL CLD
0021  0201  A9 0F                LDA #$0F          ;SETZE DATENRICHTUNGS-
0022  0203  8D 02 AC             STA DDRB          ;REGISTER AUF AUSGABE
0023  0206  A9 00                LDA #$00
0024  0208  8D 00 AC             STA IORB          ;SCHALTE RELAIS AUS
0025  020B  A5 F4         LOOP   LDA HOUR          ;HIER HAUPTSCHLEIFE
0026  020D  20 FA 82             JSR OUTBYT        ;DISPLAY ZEIGT STUNDE
0027  0210  A5 F5                LDA MIN
0028  0212  20 FA 82             JSR OUTBYT        ;DISPLAY ZEIGT MINUTE
0029  0215  20 06 89             JSR SCAND         ;DISPLAY AKTIVIEREN
0030  0218  EA                   .BYTE $EA,$EA,$EA
0030  0219  EA
0030  021A  EA
0031  021B  EA                   .BYTE $EA,$EA,$EA
0031  021C  EA
0031  021D  EA
0032  021E  EA                   .BYTE $EA,$EA,$EA
0032  021F  EA
0032  0220  EA
0033  0221  EA                   .BYTE $EA,$EA,$EA
0033  0222  EA
0033  0223  EA
0034  0224  EA                   .BYTE $EA,$EA,$EA
0034  0225  EA                                   ;   HIER KANN DER
0034  0226  EA                                   ;   BENUTZER JUMP-
0035  0227  EA                   .BYTE $EA,$EA,$EA;   BEFEHLE ZU DEN
0035  0228  EA                                   ;   UNTERPROGRAMMEN
0035  0229  EA                                   ;   EINFÜGEN, DIE
0036  022A  EA                   .BYTE $EA,$EA,$EA;   ANGESPRUNGEN WERDEN
0036  022B  EA                                   ;   SOLLEN.
0036  022C  EA
0037  022D  EA                   .BYTE $EA,$EA,$EA
0037  022E  EA
0037  022F  EA
0038  0230  EA                   .BYTE $EA,$EA,$EA
0038  0231  EA
0038  0232  EA
0039  0233  EA                   .BYTE $EA,$EA,$EA
0039  0234  EA
0039  0235  EA
0040  0236  4C 0B 02             JMP LOOP

SYMBOL TABELLE

CONTRL  0200     DDRB   AC02     HOUR   00F4
IORB    AC00     LOOP   020B     MIN    00F5
OUTBYT  82FA     SCAND  8906
```

Bild 4.38: Das Heim-Steuer-Programm (großformatiges Listing im Anhang C)

Ausgänge zu definieren (für die Relais). Natürlich sollte man nur diejenigen Leitungen als Ausgabeleitungen vereinbaren, an die auch tatsächlich Relais angeschlossen sind. Alle anderen sollten auf Eingabe gesetzt bleiben. Als eine übliche Vorkehrung ist im Programm ein Befehl vorgesehen, der die Relais explizit ausschaltet. Hierzu wird eine „0“ in das IORB (Adresse AC00) geschrieben.

Das Programm greift auf zwei Routinen des SYM-Monitors zu, um die Ausgabe der Uhrzeit einfach zu gestalten. Der Akkumulator wird mit dem Inhalt des Speichers HOUR geladen, der die aktuelle Uhrzeit in Stunden enthält (siehe Programmbeschreibung 24-Stunden-Uhr). Dann wird die Routine OUTBYT aufgerufen, die dieses Byte HOUR über das SYM-Display ausgibt. Genauso werden die Minuten aus dem Speicher MIN angezeigt, indem der Akkumulator mit MIN geladen und dann die Routine OUTBYT aufgerufen wird.

Die Routine OUTBYT steht im Monitor ab Adresse 82FA. Diese Routine zeigt den Inhalt des Akkumulators zweistellig als Hexadezimalzahl an. Dann wird die Monitorroutine SCAND (= scan display) aufgerufen. Sie steht im Monitor ab Adresse 8906. Mit ihr wird das Display eingeschaltet. Nachdem die Zeit nun angezeigt ist, werden beim Vorliegen bestimmter Bedingungen geeignete Sprungbefehle ausgeführt. Da diese natürlich von der jeweiligen Anwendung abhängen, wurden sie im Programm freigelassen und sollten erst vom Benutzer eingefügt werden. Als Übung empfehlen wir, die Relais zu einer bestimmten Zeit (2 oder 3 Minuten nach Starten des Programmes beispielsweise) einzuschalten. Das Schaltgeräusch der Relais zeigt dann, ob das Programm richtig arbeitet. Das sollte man sicherheitshalber überprüfen, bevor man dem Programm die eigentliche Gerätesteuerung überträgt.

Ein Telefonwähler

Wir werden ein Programm entwickeln, das in der Lage ist, eine einmal in die Speicher geschriebene Telefonnummer automatisch zu wählen. Bei normalen Telefonen wird von der Wählscheibe einfach eine Impulskette erzeugt. Das sollte nun schon recht einfach für uns sein und wir wollen daher ein Programm erstellen, das die Steuertöne für Tastentelefone erzeugen kann. Die in den USA verwendeten Frequenzen zeigt Bild 4.39. Das Drücken jeder Taste bewirkt die Erzeugung von zwei Tönen. Die verschiedenen Frequenzen wurden von den Fernmeldeverwaltungen sehr sorgfältig ausgewählt, um die gegenseitige Beeinflussung durch Oberwellen bei gleichzeitig kleinster Bandbreite auszuschließen. Wie das Bild zeigt, reicht der Frequenzbereich von 697 Hz bis 1477 Hz.

Unser Programm wird gleichzeitig zwei Töne erzeugen, die denselben Lautsprecher ansteuern. Damit die Tonrufauswerter diese Töne richtig erkennen, müssen die Frequenzen sehr genau eingehalten werden. Man erreicht das durch die Verwendung von zwei Zeitgebern. Wir werden die

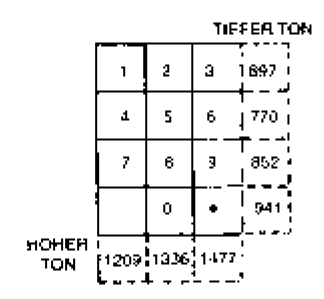


Bild 4.39: Die Telefon-Freqnenzen (USA)

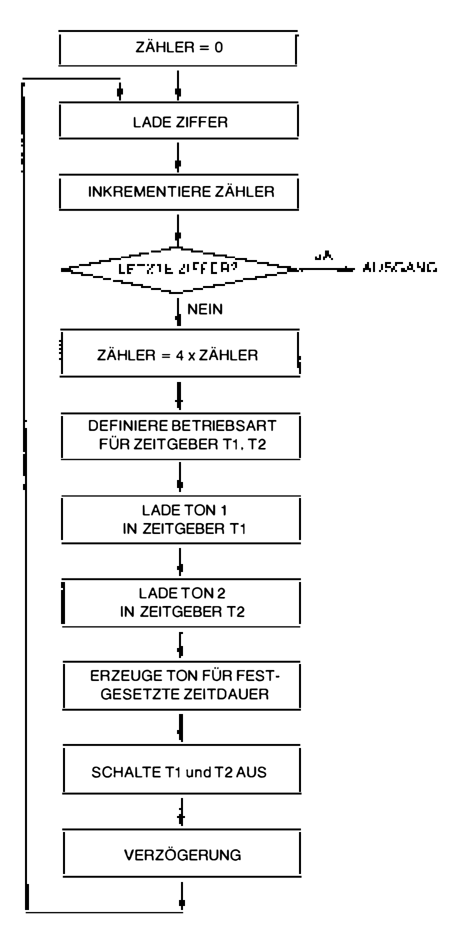


Bild 4.40: Telefonwähler, Flußdiagramm

Zeitgeber A und B auf der Mikrocomputerplatine benutzen. Jeder der beiden Zeitgeber erzeugt einen der beiden Töne und die Ausgangssignale beider Zeitgeber speisen denselben Lautsprecher. Für zuverlässigere Resultate wird allerdings die Verwendung eines Addierverstärkers (Operationsverstärker) zum Treiben des Lautsprechers dringend empfohlen. Am Programm selbst würde sich aber dadurch nichts ändern. Bild 4.40 zeigt das zugehörige Flußdiagramm. Es ist egal, wieviele Ziffern die Telefonnummer hat. Das Programm verarbeitet Telefonnummern beliebiger Länge. Die erste Nummer, die gewählt werden soll, wird aus dem Speicher geholt. Der Speicher enthält auch eine Umwandlungstabelle, die die Periodendauer beider Töne, die zur jeweiligen Nummer gehören, enthält. Genauer gesagt enthält diese Tabelle die Zeiten für eine *halbe Periode*, und da pro Ziffer zwei Töne erzeugt werden müssen, wird die Tabelle vier Bytes pro Ziffer enthalten. Die jeweilige Ziffer muß daher vervierfacht werden, um als Zähler verwendet zu werden.

Die beiden Tabellenwerte werden nacheinander aus der Tabelle in die Zählregister der Zeitgeber übertragen. Dadurch werden die Zeitgeber auch gleichzeitig gestartet. Die beiden Töne werden dann automatisch für eine bestimmte Zeit erzeugt (eine halbe oder ganze Sekunde). Dann wird ein Ruheintervall eingeschoben und nach diesem die nächste Zahl aus dem Speicher geholt. Dieser Vorgang wird so oft wiederholt, bis die ganze Nummer gewählt ist. Das Flußdiagramm ist im wesentlichen linear. Wir wollen das Programm nun untersuchen. Eine komplette Programmliste finden Sie im Bild 4.41.

```
ZEILE ADR     CODE      ZEILE

0002  0000              ;DIESES PROGRAMM KANN GESPEICHERTE TELEFONNUMMERN
0003  0000              ;SELBSTSTÄTIG WÄHLEN. ÜBER EINEN LAUTSPRECHER AM
0004  0000              ;AUSGANG ERZEUGT ES EIN ZWEITONSIGNAL. FÜR SCHALTBILD
0005  0000              ;SIEHE Z.B. FIG. 4.45. WENN DER LAUTSPRECHER DIREKT
0006  0000              ;AN DER SPRECHMUSCHEL LIEGT, AKTIVIERT DAS ZWEITON-
0007  0000              ;SIGNAL DIE WÄHLELEKTRONIK. DIE GEWÜNSCHTEN TELEFON-
0008  0000              ;NUMMERN WERDEN IRGENDWO IM SPEICHER ABGELEGT, BYTE
0009  0000              ;FÜR BYTE. DAS LETZTE BYTE MUSS EIN DF SEIN. FÜR DIE
0010  0000              ;TELEFONNUMMER 555-1212 WÜRDE BEISPIELSWEISE DIE
0011  0000              ;BYTEFOLGE 05 05 05 01 02 01 02 DF (ALLES HEX) IM
0012  0000              ;SPEICHER ABGELEGT. SPEICHERN SIE DANN DIE ANFANGS-
0013  0000              ;ADRESSE DER TELEFONNUMMER IN DER REIHENFOLGE LOW
0014  0000              ;HIGH IN DIE SPEICHER 00C0 und 00C1. SPRINGEN SIE
0015  0000              ;DANN DIESES UNTERPROGRAMM ENTWEDER VOM MONITOR
0016  0000              ;ODER VON IRGENDEINEM HAUPTPROGRAMM AN.
0017  0000              NUMPTR=$00C0              ;ZEIGER AUF BEGINN DER TELEFONNUMMER
0018  0000              ONDEL=$4D                 ;VERZÖGERUNGSKONSTANTE, WENN TÖNE
                                                  ;EINGESCHALTET SIND
0019  0000              DELCON=$FF                ;VERZÖGERUNGSKONSTANTE
0020  0000              ACR1=$A00B                ;MODUS ZEITGEBER 1
0021  0000              ACR2=$AC0B                ;MODUS ZEITGEBER 2
0022  0000              T1CH=$A005                ;ZÄHLREGISTER (HIGH) VON
                                                  ;ZEITGEBER 1
0023  0000              T1LH=$A007                ;ZWISCHENSPEICHER (HIGH) VON
                                                  ;ZEITGEBER 1
0024  0000              T1LL=$A004                ;ZWISCHENSPEICHER (LOW) VON
                                                  ;ZEITGEBER 1
0025  0000              T2CH=$AC05                ;DASSELBE FÜR ZEITGEBER 2
0026  0000              T2LH=$AC07
0027  0000              T2LL=$AC04
0028  0000              OFFDEL=$20                ;VERZÖGERUNGSKONSTANTE, WENN TÖNE
                                                  ;AUSGESCHALTET SIND
0029  0000                   *=$03CC
```

Bild 4.41: Das Programm Telefonwähler (großformatiges Listing im Anhang C)

```
0030  0300  AD 00     PHONE  LDY #$00          ;ZÄHLER FÜR Y-TE ZAHL DER TEL NUMMER
0031  0302  B1 C0     DIGIT  LDA(NUMPTR),Y     ;LADE Y-TE ZAHL
0032  0304  C8               INY
0033  0305  C9 0F            CMP #$0F          ;ENDE DER TEL NUMMER?
0034  0307  D0 01            BNE NOEND
0035  0309  60               RTS               ;RÜCKSPRUNG IN MONITOR BZW. IN DAS
0036  030A  0A EA EA  NOEND  ASL A             ;MULTIPLIZIERE ZU WÄHLENDE ZAHL MIT 4
0037  030D  0A EA EA         ASL A             ;ERGIBT INDEX FÜR FREQUENZTABELLE
                                               ;   (JEDER EINTRAG IST 4 BYTE LANG)
0038  0310  AA               TAX               ;X=INDEX FÜR TABELLENZUGRIFF
0039  0311  A9 C0            LDA #$C0
0040  0313  8D 0B A0         STA ACR1          ;BEIDE ZEITGEBER IM FREILAUFMODUS
0041  0316  8D 0B AC         STA ACR2
0042  0319  BD 50 03         LDA TABLE,X       ;LADE ERSTEN TON, LOW
0043  031C  8D 04 A0         STA T1LL          ;SPEICHERE IN ZEITGEBER 1
0044  031F  E8               INX
0045  0320  BD 50 03         LDA TABLE,X       ;LADE ERSTEN TON, HIGH
0046  0323  8D 07 A0         STA T1LH          ;SPEICHERE IN ZEITGEBER 1
0047  0326  8D 05 A0         STA T1CH          ;STARTE ZEITGEBER 1
0048  0329  E8               INX
0049  032A  BD 50 03         LDA TABLE,X       ;ZWEITER TON, LOW BYTE
0050  032D  8D 04 AC         STA T2LL          ;IN ZEITGEBER 2
0051  0330  E8               INX
0052  0331  BD 50 03         LDA TABLE,X       ;ZWEITER TON, HIGH BYTE
0053  0334  8D 07 AC         STA T2LH          ;IN ZEITGEBER 2
0054  0337  8D 05 AC         STA T2CH          ;STARTE ZEITGEBER 2
0055  033A  A2 40            LDX #ONDEL        ;LADE VERZÖGERUNGSKONSTANTE FÜR
                                               ;EINSCHALTPERIODE
0056  033C  20 55 03  ON     JSR DELAY         ;EINSCHALTVERZÖGERUNG
0057  033F  CA               DEX
0058  0340  D0 FA            BNE ON
0059  0342  A9 00            LDA #$00
0060  0344  8D 0B A0         STA ACR1          ;SCHALTE BEIDE ZEITGEBER AUS
0061  0347  8D 0B AC         STA ACR2
0062  034A  A2 20            LDX #OFFDEL       ;LADE VERZÖGERUNGSKONSTANTE FÜR
                                               ;   AUSSCHALTPERIODE
0063  034C  20 55 03  OFF    JSR DELAY         ;AUSSCHALTVERZÖGERUNG
0064  034F  CA               DEX
0065  0350  D0 FA            BNE OFF
0066  0352  4C 02 03         JMP DIGIT         ;SPRINGE AN ANFANG ZURÜCK UND
                                               ;   BEARBEITE NÄCHSTE ZAHL
0067  0355            ;
0068  0355            ;EINFACHE VERZÖGERUNGSROUTINE FÜR EIN- UND AUSSCHALTPERIODE
0069  0355            ;
0070  0355  A9 FF     DELAY  LDA #DELCON       ;LADE VERZÖGERUNGSKONSTANTE
0071  0357  38        WAIT   SEC               ;
0072  0358  E9 01            SBC #$01
0073  035A  D0 FB            BNE WAIT          ;VERZÖGERUNGSSCHLEIFE
0074  035C  60                 RTS             ;RÜCKSPRUNG AUS ROUTINE DELAY
0075  035D            ;
0076  035D            ;DIES IST DIE TABELLE DER KONSTANTEN FÜR DIE TON-
0077  035D            ;FREQUENZEN, DIE ZU DEN 10 ZIFFERN GEHÖREN. DIE KONSTANTEN
0078  035D            ;SIND JE 2 BYTE LANG, DAS LOW BYTE STEHT ZUERST.
0079  035D            ;
0080  035D  13        TABLE  .BYTE $13,$02,$76,$01   ;2 TÖNE FÜR '0'
0080  035E  02
0080  035F  76
0080  0360  01
0081  0361  C0               .BYTE $C0,$02,$9E,$01   ;'1'
0081  0362  02
0081  0363  9E
0081  0364  01
0082  0365  C0               .BYTE $C0,$02,$76,$01   ;'2'
0082  0366  02
0082  0367  76
0082  0368  01
0083  0369  C0               .BYTE $C0,$02,$53,$01   ;'3'
0083  036A  02
0083  036B  53
0083  036C  01
0084  036D  89               .BYTE $89,$02,$9E,$01   ;'4'
0084  036E  02
0084  036F  9E
0084  0370  01
0085  0371  89               .BYTE $89,$02,$76,$01   ;'5'
0085  0372  02
0085  0373  76
0085  0374  01
0086  0375  89               .BYTE $89,$02,$53,$01   ;'6'
0086  0376  02
0086  0377  53
0086  0378  01
0087  0379  48               .BYTE $48,$02,$9E,$01   ;'7'
```

Bild 4.41: Das Programm Telefonwähler (großformatiges Listing im Anhang C) Fortsetzung

```
0087  037A  02
0087  037B  9E
0087  037C  01
0088  037D  48        .BYTE $48,$02,$76,$01   ;'8'
0088  037E  02
0088  037F  76
0088  0380  01
0089  0381  48        .BYTE $48,$02,$53,$01   ;'9'
0089  0382  02
0089  0383  53
0089  0384  01
0090  0385            .END

SYMBOL TABELLE

ACR1    A00B    ACR2    AC0B    DELAY   0355    DELCON  00FF
DIGIT   0302    NOEND   030A    NUMPTR  00C0    OFF     034C
OFFDEL  0020    ON      033C    ONDEL   0040    PHONE   0300
T1CH    A005    T1LH    A007    T1LL    A004    T2CH    AC05
T2LH    AC07    T2LL    AC04    TABLE   0350    WAIT    0357
```

Bild 4.41: Das Programm Telefonwähler (großformatiges Listing im Anhang C) Fortsetzung

Das Y-Register wird als Zeiger auf die nächste zu wählende Zahl der Telefonnummer benutzt. Es wird eingangs auf Null gesetzt:

PHONE LDY #$00

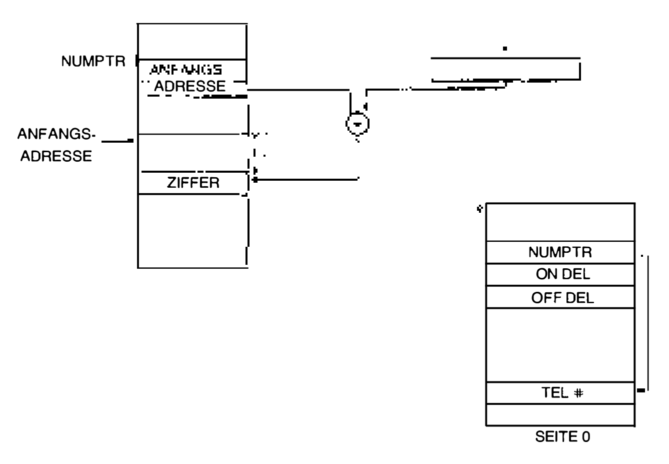


Bild 4.42: Telefonwähler: Indirekt indizierter Zugriff (oben) und Speicherbelegung (unten)

Als nächstes wird eine Ziffer der Telefonnummer geholt. Der Ladebefehl ist indirekt indiziert adressiert (siehe Bild 4.42). Wir nehmen an, daß die komplette Telefonnummer sequentiell ab der Startadresse mit dem Namen NUMPTR (engl. number pointer, d. h. Telefonnummer-Zeiger) abgelegt ist. Weiter soll die Telefonnummer mit der Zahl 0F enden, um das Ende eindeutig festzulegen.

DIGIT LDA (NUMPTR),Y HOLE NÄCHSTE ZAHL

Dann wird das Y-Register inkrementiert, so daß es auf die nächste Zahl zeigt. Wir testen, ob dies die letzte Zahl („0F“) war, und beenden das Programm folgendermaßen, falls dies der Fall war:

```
        INY
        CMP  #$0F
        BNE  NOEND
        RTS
```

Wir wollen jedoch annehmen, daß die letzte Ziffer der Telefonnummer noch nicht erreicht ist und fahren im Programm fort. Wir haben bereits erwähnt, daß die Umwandlungstabelle zwischen den Zahlen und halben Perioden vier Bytes pro Zahl umfaßt. Daher müssen wir den Wert der Zahl vervierfachen. Die Multiplikation mit vier erreichen wir durch zweifaches Linksschieben. Das Resultat schieben wir in das X-Register, so daß wir es als Index verwenden können:

```
NOEND   ASL  A          x2
        ASL  A          x2
        TAX             SCHIEBE A NACH X
```

Als nächstes werden die beiden Zeitgeber auf Freilaufmodus geschaltet:

```
        LDA  #$C0
        STA  ACR1
        STA  ACR2
```

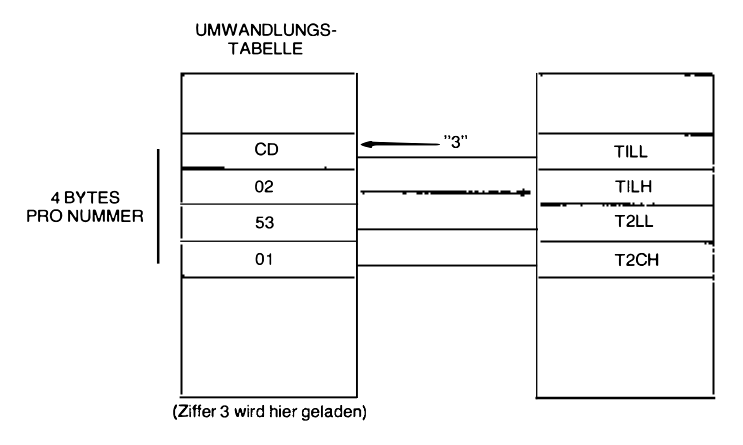


Bild 4.43: Laden des Zeitgebers

Dann werden die beiden Zählregister mit den aus der Umwandlungstabelle (Bild 4.43) entnommenen Werten für die halben Perioden geladen:

```
        LDA  TABELLE,X
        STA  T1LL
        INX
        LDA  TABELLE,X
        STA  T1LH
        STA  T1CH
        INX
        LDA  TABELLE,X
        STA  T2LL
        INX
        LDA  TABELLE,X
        STA  T2LH
        STA  T2CH
```

Sobald einmal beide Zeitgeber aktiviert sind, müssen die Töne nur noch auf eine bestimmte Dauer begrenzt werden. Die Dauer wird durch die Variable ONDEL bestimmt. Die benötigte Verzögerung erreicht man mit der Routine DELAY und einer zweiten geschachtelten Schleife „ON“.

```
        LDX  #ONDEL       DAUER DES TONS
ON      JSR  DELAY        VERZÖGERUNGS-
                          ROUTINE
        DEX               X=X-1
        BNE  ON           SCHLEIFE ON
```

Wenn nun der Ton für die nötige Zeit erzeugt wurde, werden die beiden Zeitgeber einfach wieder ausgeschaltet:

```
        LDA  #$00         „00000000“
        STA  ACR1         ZEITGEBER1 AUS
        STA  ACR2         ZEITGEBER2 AUS
```

und eine kurze Pause wird eingeschoben:

```
        LDX  #OFFDEL      DAUER DER PAUSE
OFF     JSR  DELAY        VERZÖGERUNGS-
                          ROUTINE
        DEX               X=X-1
        BNE  OFF          SCHLEIFE OFF
```

Dann springt das Programm an den Anfang zurück, wo es denselben Vorgang mit der nächsten Ziffer wiederholt:

```
                JMP   DIGIT
```

Die DELAY-Routine ist fast schon klassisch:

```
DELAY     LDA  #DELCON      LADE DAUER
WAIT      SEC
          SBC  #$01         A=A-1
          BNE  WAIT         SCHLEIFE
          RTS
```

Die Umwandlungstabelle, die die Zählerwerte für die verschiedenen Töne beinhaltet, ist am Ende des Programmes in Bild 4.41 aufgeführt.

Wir wollen nun berechnen, mit welchen Zahlen die Zählregister geladen werden müssen, um die verschiedenen Töne zu erzeugen. Es gibt sieben verschiedene Frequenzen: 697 Hz, 770 Hz, 852 Hz, 941 Hz, 1209 Hz, 1336 Hz und 1477 Hz.

Als Beispiel betrachten wir die Frequenz 697 Hz. Mit f = 697 Hz ergibt sich eine Periode von T = 1/f = 1/697 Hz = 1434,7 μsec. Die halbe Periodenlänge ist also 717 μsec oder hexadezimal 02CD.

GEWÜNSCHTE FREQUENZ	HALBE PERIODE	N = HALBE PERIODE − 1,7	HEX (für Übg. 4.4)
697	717.3	716	02CC
770	649.3	648	0288
852	586.8	585	0249
941	531.3	530	0212
1209	413.5	412	019C
1336	374.2	372	0174
1477	338.5	337	0151

Bild 4.44: Berechnung der Zeitkonstanten

Genauso berechnet man die Zeitkonstanten für die anderen Frequenzen. Bild 4.44 zeigt die Ergebnisse. Die entsprechenden hexadezimalen Werte wurden im Programm in Bild 4.41 verwendet.

Wir wollen nun einige Verbesserungen dieses Standardprogrammes untersuchen.

Übungsaufgabe 4.4: *Man kann die Frequenzgenauigkeit um einiges verbessern. Im Kapitel 2 dieses Buches oder in einem Datenblatt liest man, daß der Zeitgeber 1 im Freilaufmodus einen Ton erzeugt, der nicht genau dem erwarteten Wert entspricht. Vielmehr addiert er 1,5 oder 2 µsec zu dem im Zählregister gespeicherten Wert. Machen Sie eine Ausgleichsrechnung für die halben Perioden unter der Annahme, daß Zeitgeber 1 und 2 im Mittel 1,75 µsec zum gespeicherten Wert hinzuaddieren. Welche Werte müssen Sie abspeichern, um auf die richtigen Frequenzen zu kommen?*

Hinweis: Sehen Sie jetzt noch nicht nach – aber kontrollieren Sie nachher Ihre Ergebnisse anhand Bild 4.44.

Übungsaufgabe 4.5: *Durch Einfügen einer programmierbaren Stummschaltung läßt sich auch die Funktion des Programmes verbessern. Das ist in einigen Ländern im internationalen Telefonwählbetrieb oder innerhalb einer Telefongesellschaft zum Anwählen einer anderen Gesellschaft nützlich. Man muß dabei zuerst einige Zahlen wählen, um in die Leitung zu kommen, dann eine bestimmte Zeit warten, und dann die eigentliche Nummer wählen. Bauen Sie diesen Zusatz in das obige Programm ein!*

Eine hardwareseitige Verbesserung für sauberere Frequenzen zeigt Bild 4.45.

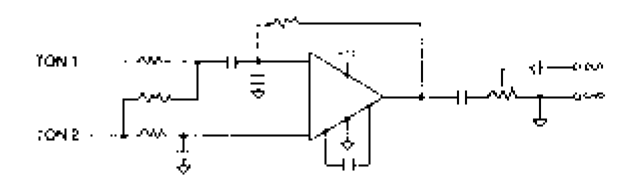


Bild 4.45: Vorschlag für hardwareseitige Verbesserung für sauberere Frequenzen

Hinweis des Übersetzers: Das Telefonnetz in den USA ist anders organisiert als das deutsche. Das beschriebene Programm läßt sich derzeit in Deutschland noch nicht einsetzen. Grundsätzlich sind bei derartigen Anwendungen die postalischen Bestimmungen zu beachten.

Abschnitt 2: Kombinierte Verfahren

Einleitung

Die in diesem Abschnitt vorgestellten Programme werden Kombinationen der bisher in diesem Kapitel vorgestellten Verfahren anwenden. Sie wurden für den KIM entwickelt. Für die folgenden Beispiele werden die einzigen Unterschiede zwischen SYM und KIM die Adressen der PIO's sein. Der interessierte Leser sei für die Speicherbelegung des KIM auf Bild 2.4 verwiesen. Da die Programme in Assembler unter Verwendung symbolischer Marken und Operanden geschrieben sind, wären die meisten von ihnen für den SYM identisch. Lediglich während der Assemblierung durch einen automatischen Assembler wie den im Anhang A beschriebenen oder bei der Kodierung von Hand, also nur an der Stelle, wo die Befehle in die endgültige hexadezimale Form gebracht werden, treten Unterschiede auf, die durch die unterschiedlichen Speicheradressen bedingt sind. Das gilt im wesentlichen natürlich auch für alle anderen 6502-Mikrocomputer, sofern diese über die benötigten E/A-Bausteine verfügen.

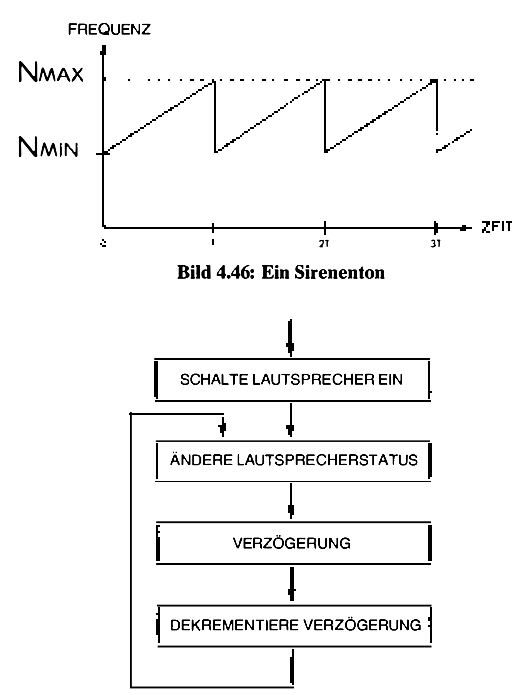


Bild 4.46: Ein Sirenenton

Bild 4.47: Flußdiagramm „Sirene" – positive Rampe

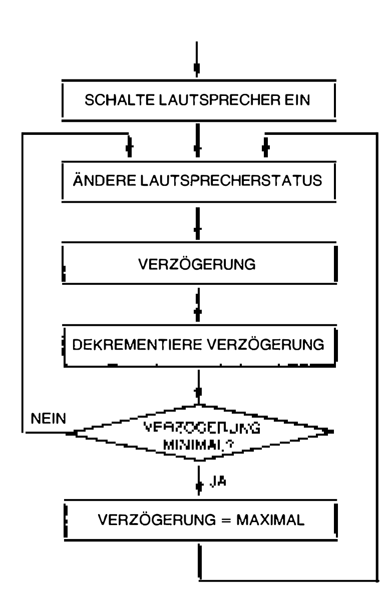


Bild 4.48: Rampenabbruch bei Nmax

Erzeugung eines Sirenentones

Bild 4.46 zeigt in einer grafischen Darstellung, was ein Sirenenton ist. Der Ton beginnt mit einer unteren Frequenz fmin und erhöht seine Frequenz während der Zeit T bis zu einem Maximalwert fmax. Die Tonfrequenz fällt dann augenblicklich auf den Startwert fmin zurück und steigt dann

```
                         ;SIREN
                         ;
                         PA        =$1700
                         PAD       =$1701
                         ;
0000: FF                 DELAY     .BYT $FF
                                   .=$40
0040: A9 01                        LDA #$01
0042: 8D 01 17                     STA PAD
0045: 8D 00 17                     STA PA
0048: EE 00 17           SWITCH    INC PA
004B: A6 00                        LDX DELAY
004D: CA                 LOOP      DEX
004E: D0 FD                        BNE LOOP
0050: C6 00                        DEC DELAY
0052: 4C 48 00                     JMP SWITCH


SYMBOL TABLE:
 PA          1700          PAD          1701          DELAY          0000
 SWITCH      0048          LOOP         004D
DONE
```

Bild 4.49: Sirenenprogramm zum Flußdiagramm in Bild 4.47

wieder bis zum Zeitpunkt 2T, usw. Das Flußdiagramm für die Erzeugung eines Tones mit ansteigender Frequenz zeigen wir in Bild 4.47. Weiterhin sollte die Frequenz einen oberen Grenzwert nicht überschreiten, da der Ton sonst unhörbar werden kann (oder der Lautsprecher gibt ihn nicht mehr wieder). Das Flußdiagramm für eine derartige durchlaufende Rampenerzeugung zeigt Bild 4.48.

Bild 4.49 zeigt das Programm. Es nähert den Verlauf von Bild 4.46 recht gut an.

Der Lautsprecher ist an das Register IORA, Bit 0 (Adresse 1700) angeschlossen. Er kann direkt mit dem Ausgang verbunden werden. Für einen lauteren Ton ist die Zusammenschaltung nach Bild 4.50 zu empfehlen. Das Datenrichtungsregister des Tores A (DDRA) für diese PIO muß so geladen werden, daß Bit 0 ein Ausgabekanal ist:

```
        LDA  #$01      „00000001“
        STA  PAD       DDRA
```

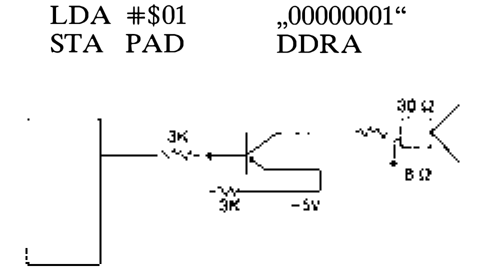


Bild 4.50: Verbesserter Anschluß eines Lautsprechers

Der Lautsprecher kann nun eingeschaltet werden. Durch einen kleinen Programmiertrick kann man den Lautsprecher ganz einfach ein- und ausschalten. Man benutzt dabei den Befehl INC. Dieser Befehl inkrementiert den Inhalt des adressierten Speichers und erzeugt daher in diesem Speicher bei mehrfacher Anwendung aufeinanderfolgende Zahlen, die abwechselnd gerade und ungerade sind. Hierdurch erreicht man, daß das niederwertigste Bit (Bit 0, also das Bit, an das unser Lautsprecher angeschlossen ist) zwischen dem Wert 1 und 0 hin- und herschaltet. Dieser Kunstgriff erlaubt das Ein- und Ausschalten des Lautsprechers mit nur einem einzigen Befehl, während man zwei benötigen würde, um ein Byte in den Akkumulator zu laden und dieses dann in das IORA zu übertragen. Schalten wir also den Lautsprecher aus. Der Lautsprecher bleibt während einer bestimmten Zeit, die durch die Konstante DELAY festgelegt ist, ausgeschaltet. Die Routine DELAY sieht folgendermaßen aus:

```
              STA  PA       INITIALISIEREN DES ORA
SWITCH        INC  PA       UMSCHALTEN DES
                            LAUTSPRECHERS
              LDX  DELAY    DAUER
LOOP          DEX
              BNE  LOOP     SCHLEIFE
```

Nach der Verwendung der Konstanten DELAY wird diese dekrementiert:

DEC DELAY

Beim nächsten Durchlauf wird die Verzögerungskonstante daher etwas kleiner sein und der Ton etwas höher. Nun muß wieder der Lautsprecher umgeschaltet werden:

JMP SWITCH

Das obige Programm erzeugt die positive Rampe des Sirenentones, wie im Flußdiagramm in Bild 4.47 gezeigt.

Übungsaufgabe 4.7: *Vervollständigen Sie das Programm entsprechend dem Flußdiagramm Bild 4.48, so daß aufeinanderfolgende positive Rampen erzeugt werden und der Ton eine richtige Sirene wird.*

Übungsaufgabe 4.8: *Schreiben Sie ein Sirenenprogramm, das eine positive Rampe, anschließend eine negative Rampe, dann wieder eine positive Rampe, usw. erzeugt.*

Erkennung eines Eingangsimpulses

In diesem Programm drücken wir auf eine Taste und das Programm mißt dann, wie lange die Taste gedrückt bleibt. Dann sollen im Lautsprecher n Piepstöne erzeugt werden, wobei n die in Sekunden ausgedrückte Zeit ist, während der die Taste gedrückt war. Der Lautsprecher wird wie im vorhergehenden Programm an Bit 0 des IORA angeschlossen. Der Einfachheit halber schließen wir den Schalter an Bit 7 des IORA an. Bild 4.51 zeigt die Verdrahtung.

Das Flußdiagramm für das Programm finden Sie in Bild 4.52. Die Zeit, während der die Taste gedrückt ist, wird in Einheiten von 0,25 sec gemessen und anschließend in Sekunden umgewandelt. Dann wird der Lautsprecher aktiviert und die Dauer ausgegeben.

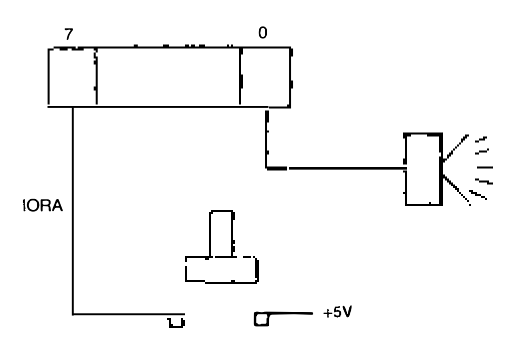


Bild 4.51: Anschluß des Lautsprechers und des Tasters

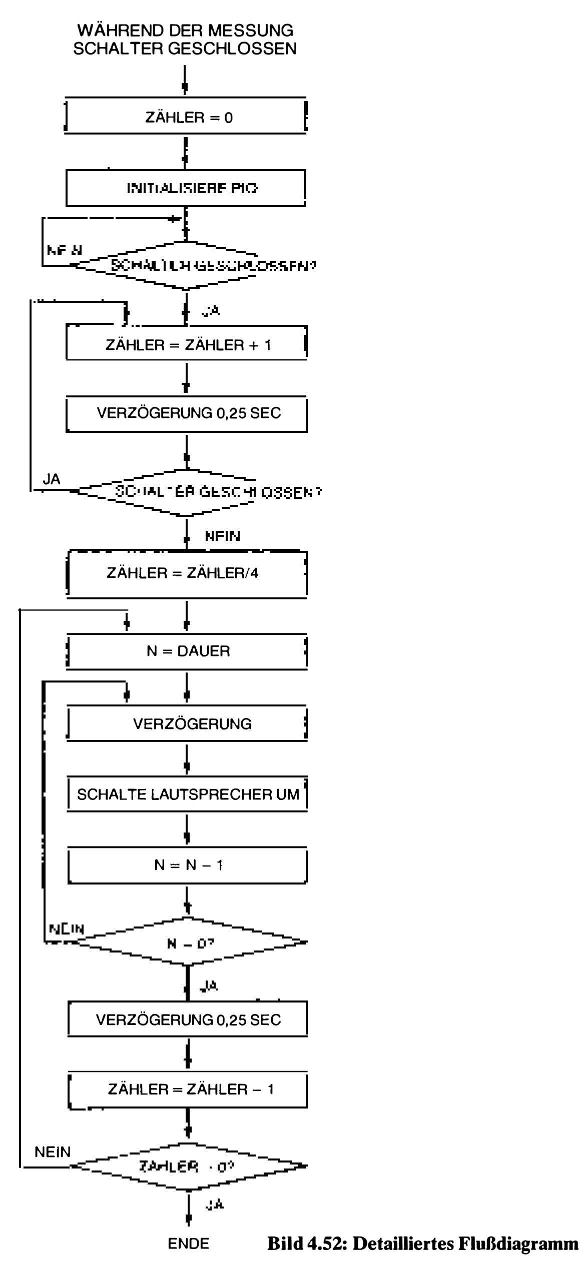


Bild 4.52: Detailliertes Flußdiagramm

Anmerkungen:
ZÄHLER enthält „n“ (Zahl der Piepser mal 4)
N ist ein Maß für eine Verzögerungsdauer

Das Programm ist in Bild 4.53 zu sehen. Es entspricht genau dem vorangestellten Flußdiagramm und dürfte sich von selbst erklären.

```
                    T       =$00
                    PA      =$1700
                    PAD     =$1701
                    ;
                            .=$40
0040: A9 01                 LDA #01
0042: 8D 01 17              STA PAD      ;PAO IST AUSGABE KANAL
0045: A9 00                 LDA #0
0047: 85 00                 STA T
0049: 8D 00 17              STA PA
004C: AD 00 17      POL     LDA PA
004F: 30 FB                 BMI POL      ;SCHALTER EIN?
0051: E6 00         CPT     INC T
0053: A2 3D                 LDX #$3D     ;0,25 SEC VERZOEGERUNG
0055: A0 00         BL2     LDY #0
0057: C8            BL1     INY
0058: D0 FD                 BNE BL1
005A: E8                    INX
005B: D0 F8                 BNE BL2
005D: AD 00 17              LDA PA
0060: 10 EF                 BPL CPT
                    ;SCHALTER EIN: LAUTSPRECHER EINMAL BETAETIGEN
0062: 46 00                 LSR T        ;DURCH VIER TEILEN
0064: 46 00                 LSR T
0066: A9 00         SOUND   LDA #0
0068: A2 80                 LDX #$80
006A: A0 00         CL2     LDY #00
006C: C8            CL1     INY
006D: D0 FD                 BNE CL1
006F: 49 01                 EOR #1
0071: 8D 00 17              STA PA
0074: E8                    INX
0075: D0 F3                 BNE CL2
                    ;ERNEUT 1/4 SEC VERZOEGERUNG
0077: A2 3D                 LDX #$3D
0079: A0 00         DL2     LDY #00
007B: C8            DL1     INY
007C: D0 FD                 BNE DL1
007E: E8                    INX
007F: D0 F8                 BNE DL2
0081: C6 00                 DEC T
0083: 10 E1                 BPL SOUND
0085: 00                    BRK


SYMBOL TABELLE
 T          0000        PA          1700        PAD         1701
 POL        004C        CPT         0051        BL2         0055
 BL1        0057        SOUND       0066        CL2         006A
 CL1        006C        DL2         0079        DL1         007B
```

Bild 4.53: Programm zur Messung von Eingangsimpulsen

Impulsmessung

Auch in diesem Programm soll die Zeit gemessen werden, während der eine Taste gedrückt ist, und anschließend ein Ton erzeugt werden. Dessen Frequenz soll der Zeit, während der die Taste gedrückt war, proportional sein.

Das Flußdiagramm zu diesem Programm ist dem vorhergehenden sehr

ähnlich. Sie finden es in Bild 4.54. Das entsprechende Programm zeigt Bild 4.55.

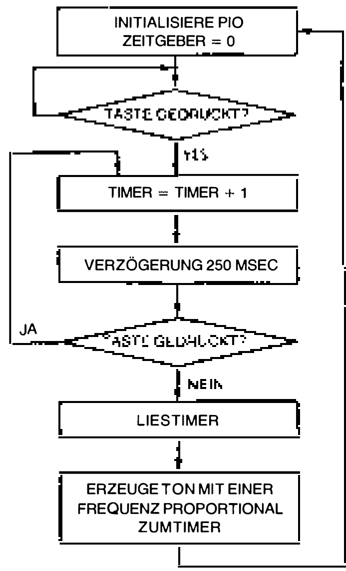


Bild 4.54: Programm zur Impulszeitmessung (Flußdiagramm)

```
                 PA       =$1700
                 PAD      =$1701
                 DL250    =$0090
                 FREQ     =$00C0
                 ;
                          .=00
0000: 00         T         .BYT $00
                 ;
                          .=$40
0040: A9 00                LDA #00
0042: 85 00                STA T        ;INITIALISIERE ZEIT
0044: 8D 00 17             STA PA
0047: A9 01                LDA #01
0049: 8D 01 17             STA PAD      ;AUS ABE BIT 0
004C: AD 00 17   POL       LDA PA       ;ABFRAGE....
004F: 30 FB                BMI POL      ;NICHT GEDRUECKT
0051: E6 00      CPT       INC T        ;INKREMENTIERE ZEIT
0053: 20 90 00             JSR DL250    ;250 MSEC VERZOEGERUNG
0056: AD 00 17             LDA PA
0059: 10 F6                BPL CPT
005B: A5 00      HERE      LDA T
005D: 0A                   ASL A        ;MULTIPLIZIERE MIT 2
005E: 0A                   ASL A        ;NOCHMAL
005F: 20 C0 00             JSR FREQ     ;ERZEUGE TON
0062: 4C 5B 00             JMP HERE

SYMBOL TABELLE
 PA          1700        PAD        1701        DL250       0090
 FREQ        00C0        T          0000        POL         004C
 CPT         0051        HERE       005B
```

Bild 4.55: Das Programm zur Impulszeitmessung

```
                          ;ERZEUGT EINEN TON, VERWENDET REG. A
                          ;NIMMT AN TOR A AUF AUSGABE GESCHALTET
                          ;FREQUENZKONSTANTE BEI AUFRUF IM REG. A
                          ;
                          PA        =$1700
                          F         =$BF
                          ;
                                    .=$C0
00C0: B5 BF               FREQ      STA F
00C2: A9 00                         LDA #0
00C4: A2 80                         LDX #$80        ;VERZOEGERUNGSKONSTANTE
00C6: A4 BF               FL2       LDY F           ;FREQUENZKONSTANTE NACH Y
00C8: C8                  FL1       INY
00C9: D0 FD                         BNE FL1
00CB: 49 01                         EOR #1
00CD: 8D 00 17                      STA PA          ;PAO UMSCHALTEN
00D0: E8                            INX
00D1: D0 F3                         BNE FL2
00D3: A5 BF                         LDA F
00D5: 60                            RTS

SYMBOL TABELLE
 PA          1700        F           00BF        FREQ        00C0
 FL2         00C6        FL1         00C8
```

Bild 4.55: Das Programm zur Impulszeitmessung (Fortsetzung)

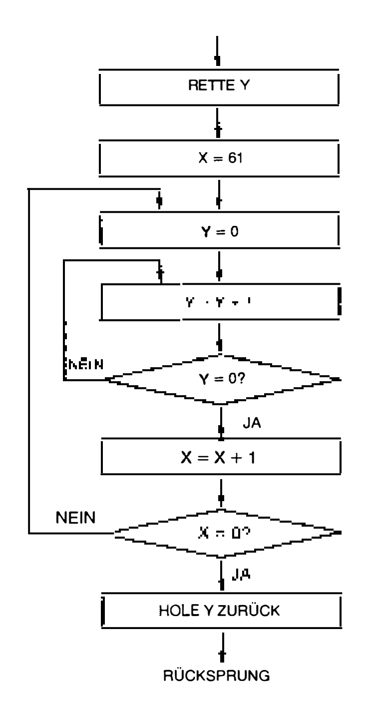


Bild 4.56: 250 msec Verzögerung, Flußdiagramm

Dieses Programm verwendet das Unterprogramm DELAY, das eine Verzögerung von 0,25 sec erzeugt. Das Flußdiagramm dieser Routine sehen Sie in Bild 4.56, das zugehörige Programm in Bild 5.57.

```
                          ;***** DL250 *****
                          ;250 MSEC VERZOEGERUNGSSCHLEIFE
                          ;Y REGISTER UNBENUTZT
                          ;
                                     =$90
0090: 98                  DL250      TYA              ;RETTE Y
0091: A2 3D                          LDX #$3D
0093: A0 00               DL2        LDY #0
0095: C8                  DL1        INY              ;INNERE SCHLEIFE
0096: D0 FD                          BNE DL1
0098: E8                             INX
0099: D0 F8                          BNE DL2          ;AEUSSERE SCHLEIFE
009B: A8                             TAY              ;Y ZURUECKHOLEN
009C: 60                             RTS

SYMBOL TABELLE
 DL250          0090          DL2          0093          DL1          0095
```

Bild 4.57: Unterprogramm DL250 (Verzögerung von 250 msec)

Übungsaufgabe 4.9: *Das Flußdiagramm in Bild 4.55 wurde so geschrieben, daß jedem Kasten im Flußdiagramm genau ein Befehl im Programm nach Bild 4.54 entspricht. Sehen Sie sich dieses Flußdiagramm oder das Programm an und schreiben Sie links neben die Anweisungen die jeweils benötigte Zahl von Taktzyklen. Berechnen Sie daraus die für die Routine insgesamt benötigte Zeit. Sind das genau 250 msec?*

Ein einfaches Musikprogramm

Als einleitenden Schritt zur Musikerzeugung wollen wir nun mit dem Lautsprecher einen Ton erzeugen, indem wir eine programmierbare Verzögerung verwenden. Bild 4.58 zeigt das Flußdiagramm und Bild 4.59 das Verzögerungsunterprogramm. Vor dem Aufruf dieses Unterprogramms muß die Konstante F mit einer passenden Verzögerungskonstanten geladen werden, die dann die Frequenz des Tones festlegt.

Um eine einigermaßen vernünftig klingende Musik zu erzeugen, muß man Töne bestimmter Frequenz und Tonlänge erzeugen können. Die Noten, mit denen die Tonlängen festgelegt werden, führen wir im folgenden auf:

Noten

(. = +50%)

♪ = 1
♩ = 2
♩. = 3
𝅗𝅥 = 4
𝅗𝅥. = 6
𝅝 = 8
𝅝. = 12

Ein Punkt, der einer Note folgt, zeigt eine um 50% längere Dauer des Tones an. Insgesamt gibt es sieben verschiedene Tonlängen. Ferner muß ein „Stummton" vereinbart werden. In *kodierter* Form benötigt man für diese Information mindestens drei Bit, oder aber in *unkodierter* Form vier Bits (unkodierte Form bedeutet, daß die Werte 1, 2, 3, 4, 6, 8, 12 durch ihr direktes binäres Äquivalent dargestellt werden).

Um alle Noten einer Oktave darzustellen, müssen wir die Noten A, B, C, D, E, F und G sowie die sechs halben Noten dazwischen vorsehen. Das ergibt pro Oktave 13 Noten. Soll mehr als eine Oktave dargestellt werden, dann sollte ein ganzes Byte pro Note verwendet werden. Falls der Leser bei seinem Mikrocomputer nur über wenig Speicherplatz verfügt, wird er den Wunsch haben, sich auf nur 16 Noten zu beschränken und ist dann in der Lage, eine Kodierung zu verwenden, bei der die linke Hälfte jedes Bytes die Dauer und die rechte Hälfte die eigentliche Note repräsentiert.

Wir werden in unserem Beispiel nur einfache Melodien spielen und eine direkte Kodierung anwenden, wobei der Tonlänge ein ganzes Byte und ebenfalls der Tonfrequenz ein ganzes Byte zugewiesen wird. Die Bilder 4.60 bis 4.62 zeigen drei Beispiele: eine Sonate von Mozart, einen Choral von Bach und ein bekanntes Kinderlied.

Das zum entsprechenden Musikprogramm gehörende Flußdiagramm finden Sie in Bild 4.63 und das vollständige Programm in Bild 4.64.

Die Bilder 4.58 und 4.59, die dem Programm vorangestellt sind, zeigen das Flußdiagramm und die Programmliste eines Programmes, das eine

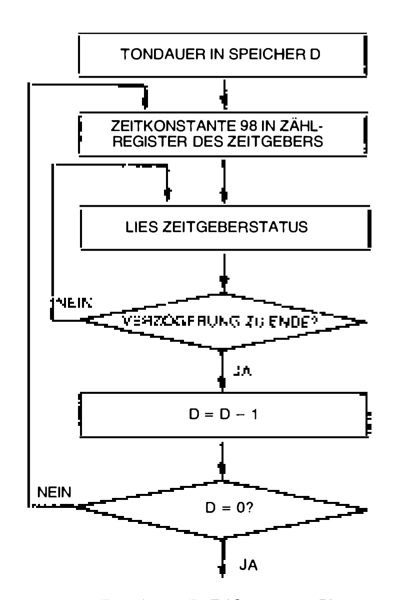


Bild 4.58: Flußdiagramm der Routine TIME10 (erzeugt Verzögerung von 0,1 sec)

Verzögerung von 0,1 sec erzeugt. Dieses Programm kommt im Programm Musik als Unterprogramm (hier allerdings mit 0,2 sec) vor.

```
                    ;***** TIME10 *****
                    ;1/10 SEKUNDE VERZOEGERUNG
                    ;
                    TIMER     =$1707
                    D         =$9D
                    ;
                             .=$9E
009E: 86 9D         TIME10    STX D
00A0: A9 62         TO        LDA #$62     ;DEZIMAL 98
00A2: 8D 07 17                STA TIMER
00A5: AD 07 17      T1        LDA TIMER
00A8: 10 FB                   BPL T1
00AA: C6 9D                   DEC D
00AC: D0 F2                   BNE TO
00AE: 60                      RTS

SYMBOL TABELLE
 TIMER        1707        D          009D        TIME10       009E
 TO           00A0        T1         00A5
```

Bild 4.59: Routine TIME10 – Erzeugung einer Verzögerung von 0,1 sec

Adresse	Dauer	F	Note	
00	09	20	*la* 𝅗𝅥	A
2	04	4F	*do#* ♩	C#
4	04	6B	*mi* ♩	E
6	05	12	sol# ♩.	G#
8	01	20	*la* 𝅘𝅥𝅯	A
A	01	39	*si* 𝅘𝅥𝅯	B
C	0F	20	*la* 𝅝	A
E	02	00		
12	09	7C	*fa#* 𝅗𝅥	F#
12	04	6B	*mi* ♩	E
14	04	91	*la* ♩	A
16	04	6B	*mi* ♩	E
18	04	59	*ré* ♩	D
1A	09	4F	*do#* 𝅗𝅥	C#
1C	00	00		
1E				
20				

Bild 4.60: Sonate von Mozart

Adresse	Dauer	F	Note
00	88	44	*do* C
02	06	59	*ré* D
04	06	6B	*mi* E
06	88	83	*sol* G
08	06	74	*fa* F
A	06	74	*fa* F
C	88	91	*la* A
E	06	83	*sol* G
10	06	83	*sol* G
12	88	A3	*do* C
14	06	9E	*si* B
16	06	A3	*do* C
18	06	83	*sol* G
1A	06	6B	*mi* E
1C	06	44	*do* C
1E	88	59	*ré* D
20	06	6B	*mi* E
22	06	20	*la* A
24	88	83	*sol* G
26	06	74	*fa* F
28	06	6B	*mi* E
2A	88	59	*ré* D
2C	06	44	*do* C
2E	06	04	*sol* G
30	06	44	*do* C
32	88	39	*si* B
34	06	44	*do* C
36	06	6B	*mi* E
38	06	83	*sol* G
3A	0E	A3	*do* C
3C	0E	44	*do* C
3E	00	00	

Bild 4.61: Choral von Bach

Übungsaufgabe 4.10: *Überprüfen Sie, ob die Routine TIME10 (oder TIME20) wirklich eine Verzögerung von genau 0,1 sec (0,2 sec) erzeugt. Berücksichtigen Sie die Dauer jedes Befehls und die Anzahl der Schleifendurchläufe. Berechnen Sie die genaue Verzögerung.*

Adresse	Dauer	F	Note
0	04	44	do ♪ C
2	04	44	do ♪ C
4	04	44	do ♪ C
6	04	59	ré ♪ D
8	09	6B	mi ♩ E
A	09	59	ré ♩ D
C	04	44	do ♪ C
E	04	6B	mi ♪ E
10	04	59	ré ♪ D
12	04	59	ré ♪ D
14	09	44	do ♩ C
16	10	00	
18	04	44	do ♪ C
1A	04	44	do ♪ C
1C	04	44	do ♪ C
1E	04	59	ré ♪ D
20	09	6B	mi ♩ E
22	09	59	ré ♩ D
24	04	44	do ♪ C
26	04	6B	mi ♪ E
28	04	59	ré ♪ D
2A	04	59	ré ♪ D
2C	09	44	do ♩ C
2E	00	00	

Bild 4.62: Kinderlied „Au clair de la lune“

Verkehrssteuerung mit KIM

Eine Möglichkeit für die Verkabelung einer Ampelanlage zur simulierten Verkehrssteuerung zeigt Bild 4.66. Die Anlage ist in allen Richtungen mit Schaltern ausgerüstet, die die Anwesenheit von Autos oder Fußgängern feststellen sollen.

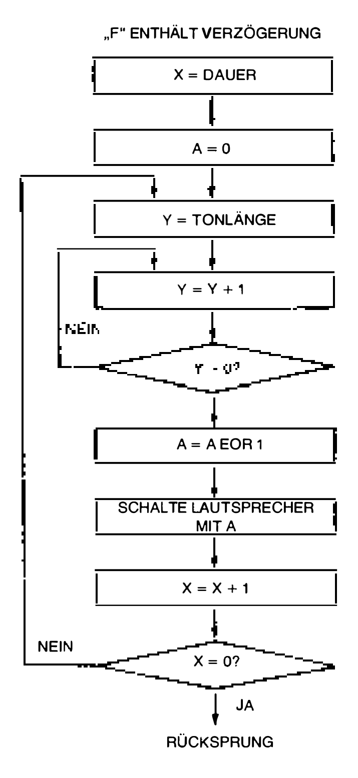


Bild 4.63: Flußdiagramm zum Musikprogramm

```
                   ;**** MUSIKPROGRAMM *****
                   PA        =$1700
                   PAD       =$1701
                   TIMER     =$1707
                             .=00
                   ADDRS     .=.+2
                   TEMP      .=.+1
                   YSAVE     .=.+1
                   F         .=.+1
                   ;
                             .=$10
0010: A9 31        TIME20    LDA #$31
0012: 8D 07 17               STA TIMER
0015: 2C 07 17     T1        BIT TIMER
0018: 10 FB                  BPL T1
001A: CA                     DEX
001B: D0 F3                  BNE TIME20
001D: 60                     RTS
```

Bild 4.64: das Musikprogramm

```
                         ;
                                  .=$20
0020: 84 03              FREQT    STY YSAVE
0022: 85 04                       STA F
0024: A9 31              FT0      LDA #$31
0026: 8D 07 17                    STA TIMER     ;STARTE ZEITGEBER (1/20 SEC)
0029: A4 04              FT1      LDY F
002B: C8                 FT2      INY
002C: D0 FD                       BNE FT2
002E: EE 00 17                    INC PA        ;SCHALTE LAUTSPRECHER UM
0031: 2C 07 17                    BIT TIMER     ;ZEIT ABGELAUFEN?
0034: 10 F3                       BPL FT1       ;NEIN: WEITERMACHEN
0036: CA                 FT3      DEX
0037: D0 EB                       BNE FT0
0039: A4 03                       LDY YSAVE
003B: 60                          RTS
                         ;
                                  .=$40
0040: A2 0F              START    LDX #$0F
0042: 9A                          TXS
0043: A9 00                       LDA #$00
0045: 8D FA 17                    STA $17FA
0048: 8D FE 17                    STA $17FE
004B: A9 1C                       LDA #$1C
004D: 8D FB 17                    STA $17FB
0050: 8D FF 17                    STA $17FF     ;INTERRUPT VEKTOR
0053: A9 01                       LDA #$01
0055: 8D 01 17                    STA PAD       ;PAO IST AUSGABELEITUNG
0058: A0 00              DACAPO   LDY #$00
005A: B1 00              NEXT     LDA (ADDRS),Y
005C: 85 02                       STA TEMP
005E: 29 7F                       AND #$7F
0060: AA                          TAX           ;DAUER
0061: F0 F5                       BEQ DACAPO
0063: C8                          INY
0064: B1 00                       LDA (ADDRS),Y
0066: F0 10                       BEQ TONE
0068: 20 20 00                    JSR FREQT
006B: 24 02                       BIT TEMP
006D: 30 05                       BMI AFTER
006F: A2 02                       LDX #$02
0071: 20 10 00                    JSR TIME20
0074: C8                 AFTER    INY
0075: AC 50 00                    JMP NEXT
0078: 20 10 00           TONE     JSR TIME20
007B: F0 F7                       BEQ AFTER

SYMBOL TABELLE
 PA          1700          PAD         1701          TIMER       1707
 ADDRS       0000          TEMP        0002          YSAVE       0003
 F           0004          TIME20      0010          T1          0015
 FREQT       0020          FT0         0024          FT1         0029
 FT2         002B          FT3         0036          START       0040
 DACAPO      0058          NEXT        005A          AFTER       0074
 TONE        0078
```

Bild 4.64: Das Musikprogramm (Fortsetzung)

Übungsaufgabe 4.11: *Schreiben Sie ein Programm zur Verkehrssteuerung, das den folgenden Anforderungen genügt:*

- *Mindestdauer der Gelbphase: 3 Sekunden*

• *Wenn ein Auto registriert wird (durch Drücken eines der Schalter) soll die aktuelle Grünphase um 3 Sekunden verlängert werden.*
• *Maximaldauer der Grünphase in allen Richtungen sei 2 Minuten, falls eine Grünanforderung (Auto) in der jeweiligen Gegenrichtung vorliegt.*
• *Nachts sollen die Ampeln blinken (Die Tag/Nacht-Anzeige wird mit einem separaten Schalter durchgeführt).*
• *Ein mögliches Flußdiagramm zeigt Bild 4.65. Schreiben Sie das zugehörige Programm.*

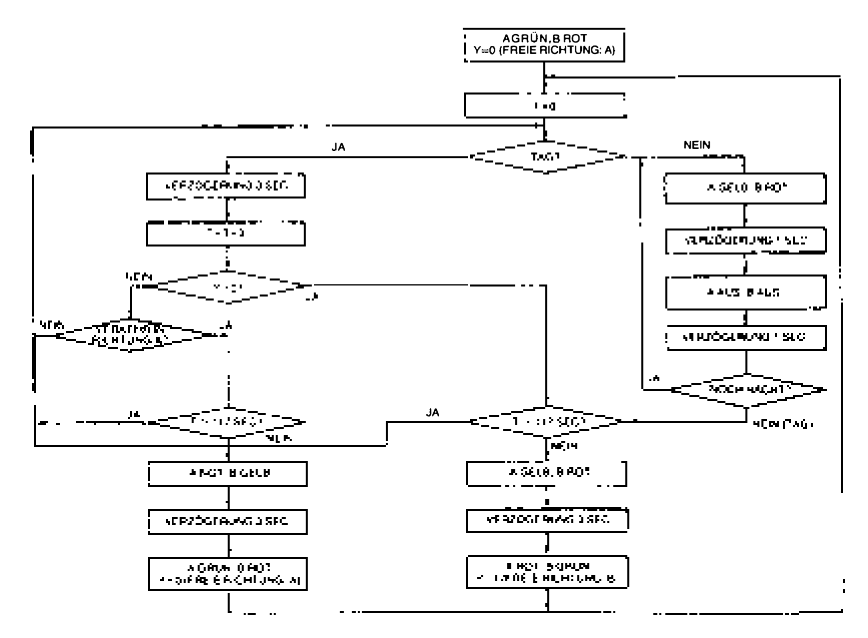


Bild 4.65: Flußdiagramm zum Programm Verkehrssteuerung

Kleines Einmaleins

Als letzte Übung soll dieses Programm das kleine Einmaleins lehren. Das Programm soll n mal mit einer LED blinken (oder den Lautsprecher kurz ansteuern), wobei n zwischen 1 und 10 liegen kann. Dann soll es 2 Sekunden warten und nochmals p Sekunden blinken, p ebenfalls zwischen 1 und 10.

Dann soll der Benutzer n x p mal auf eine Taste drücken, um seine Antwort einzugeben. Mit dem Lautsprecher sollte ein akustisches Bestätigungssignal gegeben werden. Der Benutzer beendet die Eingabe, indem er für 3 Sekunden keine Taste mehr drückt. Wenn die Antwort richtig war, soll die LED für fünf Sekunden aufleuchten, war die Antwort falsch, dann soll sie blinken.

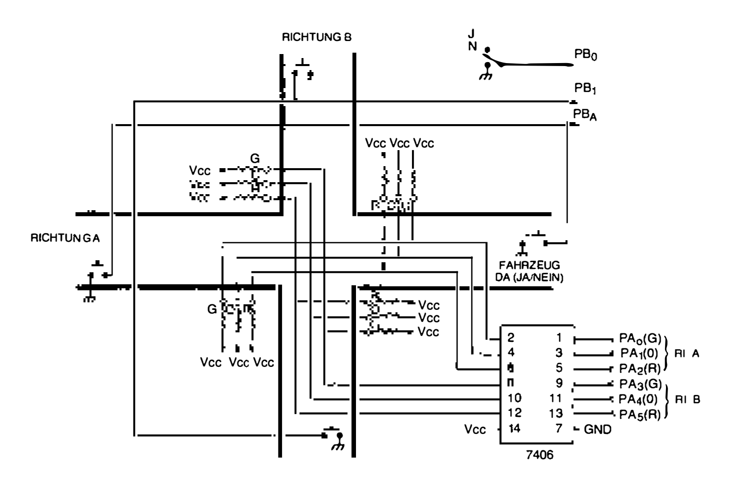


Bild 4.66: Verdrahtung der simulierten Ampelanlage

Übungsaufgabe 4.12: *Entwerfen Sie zu diesem Programm ein Flußdiagramm (Das Programm ist zwar einfach, aber etwas länger als die vorhergehenden. Falls Sie wirklich keine Lösung herausfinden, können Sie im Anhang B eine mögliche Lösung nachsehen).*

Zusammenfassung

Wir haben in diesem Kapitel einfache Ein/Ausgabegeräte an einen 6502 Mikrocomputer angeschlossen. Wir haben gelernt, wie man einfache Hardwareinterfaces realisiert und wie man einfache Benutzersoftware entwickelt, die externe Geräte oder Anlagen abfragen und steuern kann. Obwohl die hier vorgestellten Anwendungen bewußt recht einfach gewählt wurden, können mit derselben einfachen Hardware auch wesentlich komplexere Anwendungen entwickelt werden. Wir haben nun einen Stand erreicht, wo wir zu umfangreicheren Programmen und Interfaces des Kapitels 5 übergehen können, zu Industrie- und Heimanwendungen.

Kapitel 5

Anwendungen für Industrie und Heim

Einleitung

Im Kapitel 4 haben wir uns grundlegende Fertigkeiten beim Anschließen einfacher Geräte an eine 6502 Mikrocomputerplatine und bei der Entwicklung von Standardsoftware für Anwender angeeignet. In diesem Kapitel werden wir komplexere Geräte an die 6502-Platine anschließen und eine komplexere Anwendersoftware wird hierzu entwickelt werden. Die vorgestellten Anwendungen sind typisch für Steuerungen in industrieller Umgebung und auch im Haus. Im nächsten Kapitel werden wir Mikrocomputer-Peripheriegeräte an unsere Mikrocomputerplatine anschließen.

Als erste Anwendung wird eine simulierte Verkehrssteuerung vorgestellt. Die Ampelanlagen werden durch LED's auf der Platine simuliert und wir werden Benutzerprogramme steigender Komplexität entwickeln. Mit simulierten Induktionsschleifen, die normalerweise im Straßenbelag eingelassen sind, und die wir hier durch Schalter ersetzen, werden wir ankommende Autos registrieren. Die Fertigkeiten, die wir bei der Entwicklung dieser Hardware- und Softwareinterfaces benötigen, sind dieselben, wie sie für echte Industriesteueranlagen gebraucht werden.

Als nächstes werden wir eine 5x7 Punktmatrix aus LED's an das System anschließen. Dies ist bei der Anzeige von Daten eine häufig verwendete Technik. Die Darstellung mit einer Punktmatrix verwendet man nicht nur bei LED's, sondern auch, um Zeichen auf einem Bildschirm oder auf einem Matrixdrucker darzustellen. Wir werden unsere Punktmatrix dazu benutzen, die Werte von Schaltern, die vom Mikrocomputer erfaßt werden, anzuzeigen.

Dann werden wir mit dem Lautsprecher Töne erzeugen, um einfache Musikprogramme zu entwickeln. Dabei wird ein Satz von Schaltern festlegen, welche Note gespielt werden soll. Die bei der Erzeugung von Tönen mit dem Lautsprecher gewonnenen Erfahrungen werden wir im nächsten Programm dazu benutzen, Geräusche wie etwa eine Sirene darzustellen.

Im nächsten Anwenderprogramm werden wir für unser Haus oder andere Gebäude eine Einbrecher-Alarmanlage entwickeln. Eines der Geräte, die einen möglichen Einbrecher registrieren werden, wird eine Lichtschranke sein. Wird die Lichtschranke durchbrochen, dann heult ein Alarm aus dem Lautsprecher. Viele zusätzliche Verbesserungen werden in den Übungsaufgaben vorgeschlagen.

Dann werden wir die Geschwindigkeit eines normalen Gleichstrommotors vom Computer steuern lassen. Bei Verwendung digitaler Techniken ist die Steuerung von Motordrehzahlen in der Tat recht einfach. Die hierzu nötigen Techniken und Hardwareinterfaces werden vorgestellt.

In der nächsten Anwendung wird ein Temperaturfühler an die Mikrocomputerplatine angeschlossen und die gemessene Temperatur wird in Form eines Tones ausgegeben. Der Ton hat dabei eine umso höhere Frequenz, je höher die Temperatur ist. Das wird das erste Beispiel einer Analog-Digital-Wandlung sein und wir werden die für eine solche A/D-Wandlung nötigen Hardware- und Softwaretechniken vorstellen.

Dem Leser sei dringend empfohlen, die für die Programme in diesem Kapitel benötigte Benutzerplatine Nummer 2 aufzubauen. Alle auf dieser Platine verwendeten Bauteile sind billig und normalerweise bei jedem Elektronikhändler problemlos zu beziehen. Lediglich der Digital-Ana-

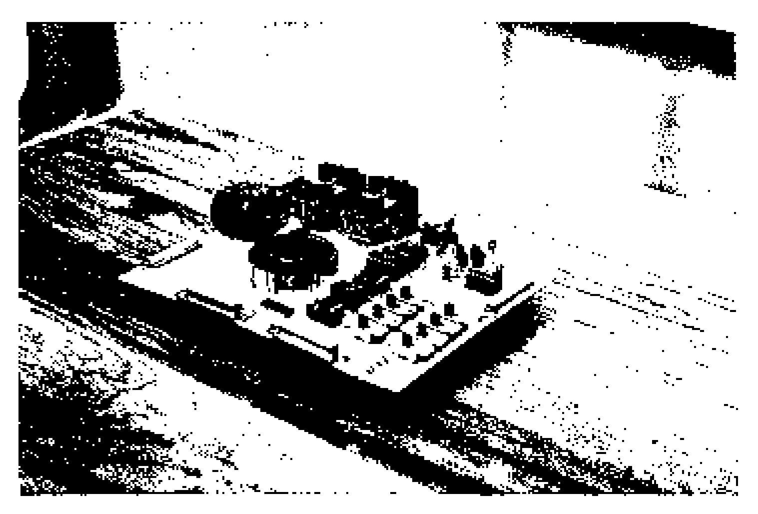


Bild 5.1: Die Benutzerplatine #2

log-Wandler muß meist vom Distributor bestellt werden. Die Bilder 5.1 bis 5.3 zeigen die Benutzerplatine.

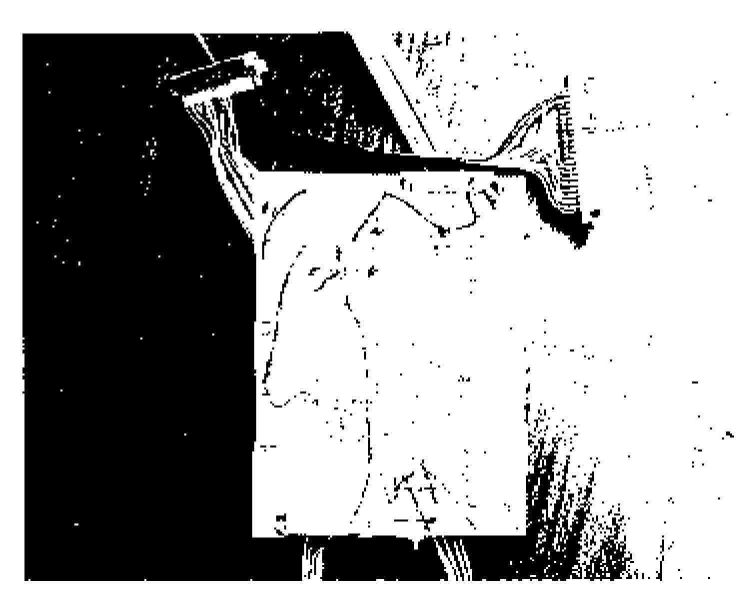


Bild 5.2: Blick auf die Verdrahtungsseite

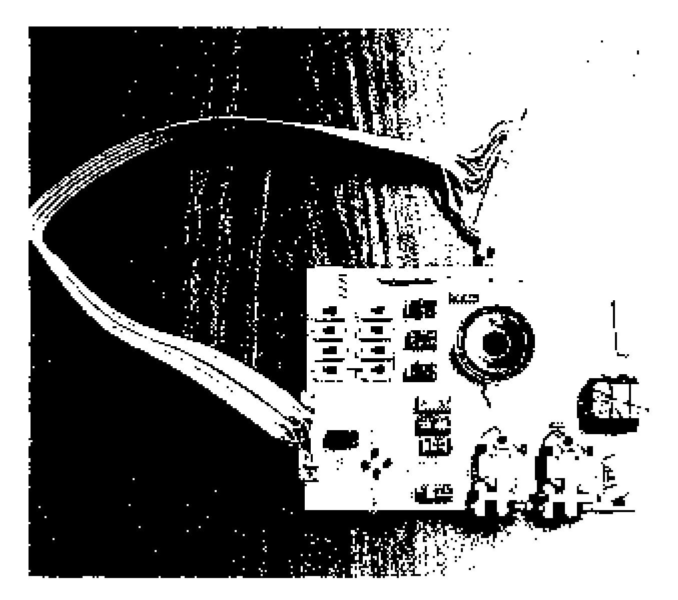


Bild 5.3: Bequemer Anschluß der Benutzerplatine an den Computer mit Steckverbindungen

Wegen der an der Ausgangsseite des Mikrocomputers normalerweise nur kleinen verfügbaren Zahl von Toren wurden vier Stecker H1, H2, H3 und H4 auf der Platine installiert, die einen bequemen Anschluß ermöglichen, ohne daß die Verdrahtung für jedes Programm neu durchgeführt werden müßte. Diese Steckverbindungen wurden so entworfen, daß sie direkt an die Kontaktleisten des SYM angeschlossen werden können, aber auch problemlos an die Ausgänge anderer Mikrocomputer angeschlossen werden können. Bei jeder Anwendung werden wir eine oder zwei Ausgangsleitungen von der Mikrocomputerplatine an den entsprechenden Stecker unserer Benutzerplatine anschließen. Die Einzelheiten der Verdrahtung werden vor jeder Anwendung erläutert.

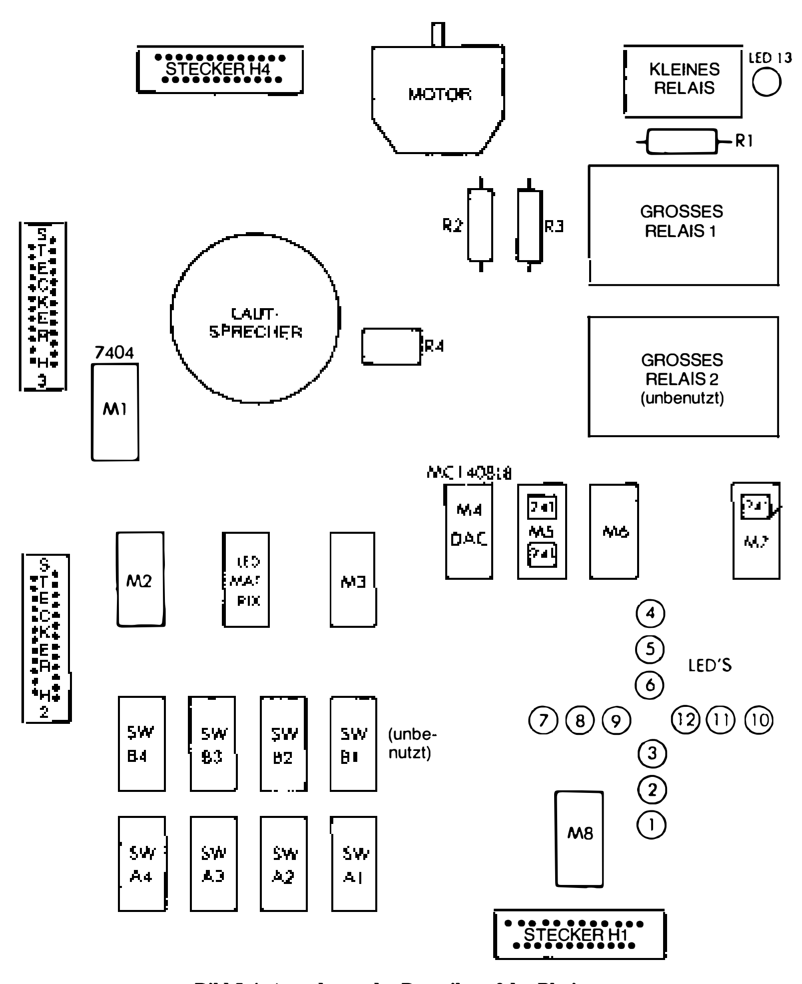


Bild 5.4: Anordnung der Bauteile auf der Platine

Die Anordnung der Bauteile auf der Platine sehen Sie in Bild 5.4. Die Verdrahtung der Steckverbindungen sehen Sie in den Bildern 5.5A und 5.5B. Die Einzelheiten der Verdrahtung werden aber in dem Abschnitt, in dem die jeweilige Anwendung besprochen wird, nochmals genauer erklärt.

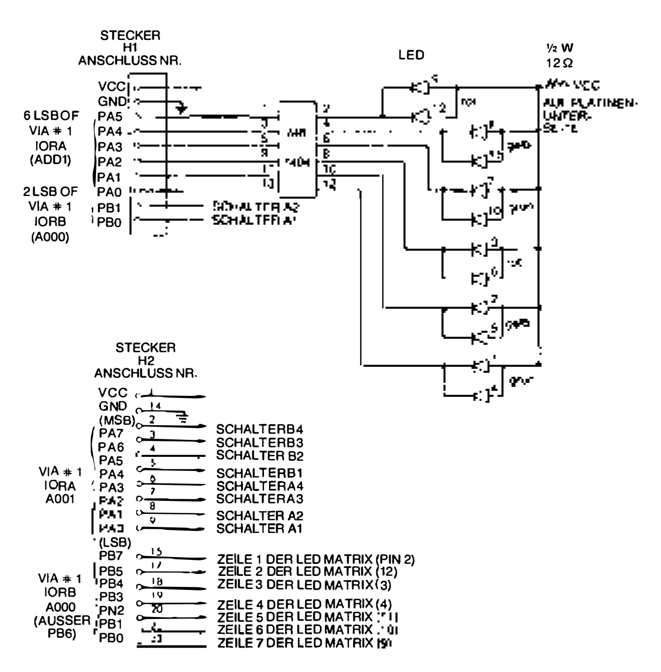


Bild 5.5A: Stecker H1 und H2

Die Bauteile auf der Platine werden auf der Unterseite nach Bild 5.2 freiliegend verdrahtet (wire-wrap Technik). Man kann die Verbindungen natürlich auch löten. Vergessen Sie nicht, die üblichen Vorsichtsmaßnahmen beim Verdrahten von hochintegrierten Schaltungen zu beachten: alle Werkzeuge und insbesondere Sie selbst sollten geerdet sein. Beachten Sie auch, daß der Trimmer (Einstellwiderstand) in Serie zum Lautsprecher nicht ganz auf Null gedreht sein darf. Falls er doch auf Null zurückgedreht ist, wird er beim Einschalten der Stromversorgung durchbrennen, wenn der Lautsprecher wie beim SYM an einen gepufferten Ausgang angeschlossen ist (außerdem wird höchstwahrscheinlich auch der Ausgangstransistor durchbrennen). Aus diesem Grund empfiehlt es sich, zusätzlich einen Festwiderstand in Reihe zu schalten.

Bei Verwendung von Computern mit weniger Toren als beim SYM (z. B. PET/CBM oder VC20) wird man die Steckerbelegung natürlich anders

wählen. Auf der Softwareseite ergeben sich hierdurch aber lediglich bei den PIO-Adressen Änderungen. Einige der beschriebenen Anwendungen bedingen jedoch mehr als 8 E/A-Leitungen, also mindestens 2 Tore.

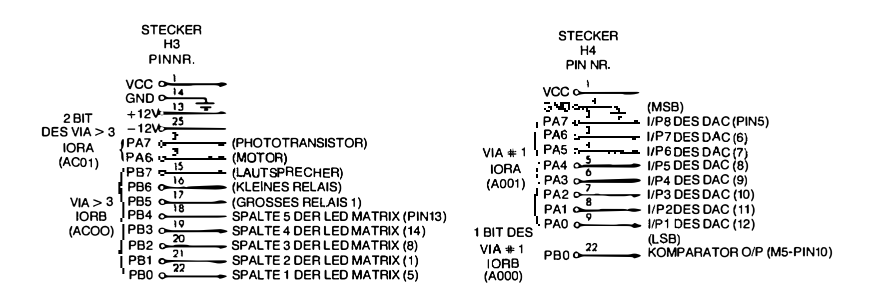


Bild 5.5B: Die Stecker H3 und H4

Das Ziel dieses Kapitels ist es, Ihnen die prinzipiellen Methoden der Anwenderprogramme näherzubringen und Sie damit in die Lage zu versetzen, Heimanwendungen schon recht bedeutender Komplexität zu entwickeln oder industrielle Steuerungsprobleme zu lösen. Am Ende dieses Kapitels sollten Sie über alle die Fertigkeiten verfügen, die Sie benötigen werden, um selbst mit der Entwicklung komplexer Anwendungen zu beginnen. Sollten Ihnen hierbei spezifische Interfaceprobleme begegnen, empfehlen wir Ihnen das Studium unseres Buches „Mikroprozessor Interface Techniken“.

Wichtiger Hinweis: Um eine Ein/Ausgabeleitung mehr benutzen zu können, ist Transistor 1 (Mitte) der vier gepufferten Ausgabeleitungen des SYM kurzgeschlossen.

Die hier vorgestellten Programme wurden so geschrieben, daß man sie noch verbessern kann. Der aufmerksame Leser wird feststellen, daß der Programmierstil oft noch verbessert werden kann. Solche Verbesserungen werden am Ende jeder Anwendung im Übungsteil vorgeschlagen oder beschrieben. Beim Lesen der Programme sollten Sie schon auf solche möglichen Verbesserungen achten. Wir werden jedoch erst im nächsten Kapitel optimierte Programme vorstellen, wenn alle anderen Probleme gelöst sind.

Auch in diesem Kapitel werden wieder viele Übungsaufgaben vorgeschlagen. Es wird dringend empfohlen, die meisten dieser Übungsaufgaben entweder auf dem Papier oder auf einem richtigen Mikrocomputer zu lösen. Sie wurden sehr sorgfältig so entworfen, daß sie sicherstellen, daß der Inhalt des vorangegangenen Abschnitts wirklich verstanden wurde und daß der Leser ihn selbständig und kreativ einsetzen kann. Wenn Sie die Übungsaufgabe lösen können, ohne nachschlagen zu müssen, dann haben Sie gelernt, ihre eigenen Anwenderprobleme effektiv zu lösen.

Eine Verkehrssteueranlage

Wir wollen ein Programm zur Verkehrssteuerung an einer simulierten Kreuzung entwickeln. Ein Diagramm der Kreuzung zeigt Bild 5.6. Der Verkehrsfluß hat zwei Richtungen, die wir A und B nennen. Im Verkehrssteuer-Jargon werden sie „Phasen“ genannt. Die beiden Ampeln für die beiden Gegenrichtungen einer Phase, beispielsweise die beiden Ampeln der A-Straße, sollen zur selben Zeit die selbe Farbe zeigen (grün, gelb, rot). Genauso sollen die beiden anderen Ampeln der Phase B ebenfalls zur selben Zeit schalten. Wir werden diese vier Ampeln auf unserer Platine mit vier LED-Sätzen (grüne, gelbe und rote LED's) simulieren. Schließlich wollen wir noch annehmen, daß im Straßenbelag vier Induktionsschleifen an den Stellen A1, A2, B1, B2 eingelassen sind (Bild 5.6). Ihre Funktion wird später genauer erläutert.

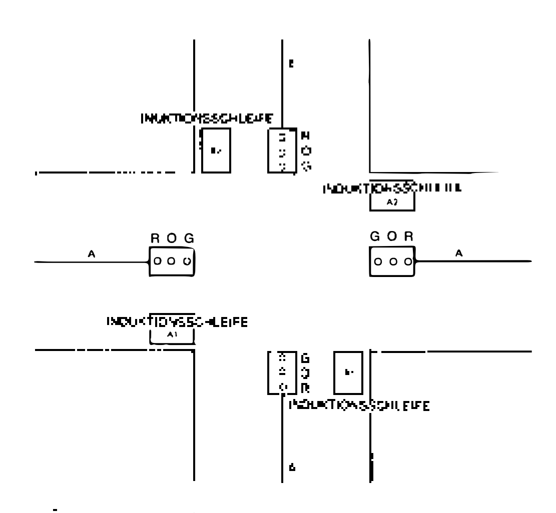


Bild 5.6: Das Verkehrssteuersystem

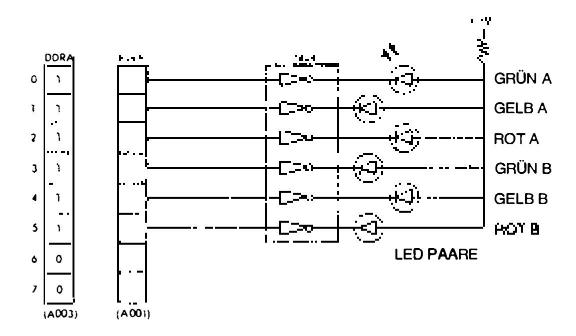


Bild 5.7: Anschluß der LED's

Zuerst wollen wir den hardwareseitigen Anschluß unserer „Ampeln“ (also unserer LED’s) an das Mikroprozessorsystem untersuchen. Nach Bild 5.7 schließen wir einen 7404 Treiber an das IORA des 6522 #1 an. Die LED-Paare sehen wir rechts im Bild. Zur besseren Übersichtlichkeit wurde nur eine LED pro Leitung eingezeichnet. Da es für jede Phase zwei Ampeln gibt, werden in Wirklichkeit aber zwei LED’s parallel geschaltet. Die tatsächliche Verdrahtung zeigt Bild 5.8. Damit die ersten 6 Bits des IORA als Ausgabekanäle festgelegt sind, wird das Datenrichtungsregister DDRA links vom IORA in Bild 5.7 mit dem entsprechenden Bitmuster „00111111“ geladen. Um genug Strom zur Beleuchtung der LED’s zu ziehen, ist ein Treiber (der 7404) nötig.

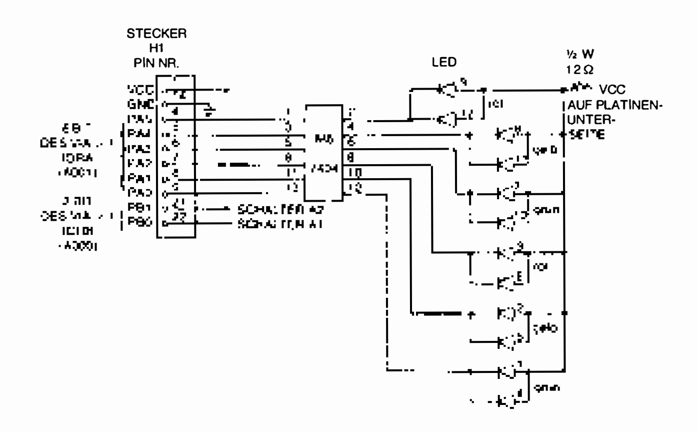


Bild 5.8: Verdrahtung aller LED’s

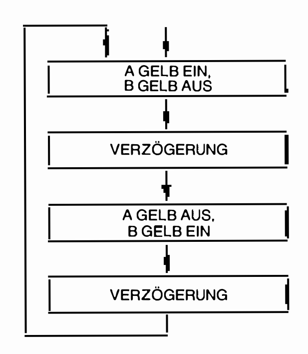


Bild 5.9: Nachtsteuerung

Wir gehen nun daran, die Programme für verschiedene Verkehrssteueralgorithmen zu entwickeln. Man kann generell zwei Fälle unterscheiden: die Nachtsteuerung (blinkende Lichter) und die Tagsteuerung.

```
0100  A9 3F        NACHT  LDA #$3F     „00111111“
0102  8D 03  A0           STA $A003    Setze VIA#1 DDRA = 3F
                                       für Ausgabemodus
0105  A9 02        NACHT2 LDA #$02
0107  8D 01  A0           STA $A001    Erste Richtung gelb Ein
010A  A9 FC               LDA $FC
010C  85 00               STA $00      Zähler für DLYA = FC
                                       (= -4) in Speicher $00
010E  20 20  01           JSR DLYA     Unterprogrammaufruf
0111  A9 10               LDA #$10
0113  8D 01  A0           STA $A001    Andere Richtung gelb Ein
0116  A9 FC               LDA #$FC
0118  85 00               STA $00      Zähler für DLYA = FC
                                       in Speicher $00
011A  20 20  01           JSR DLYA     Unterprogrammaufruf
011D  4C 05  01           JMP NACHT2   Von vorne
```

(Steckerleiste A an Stecker H1 anschließen)

Subroutine DLYA: Dieses Unterprogramm holt einen Zähler aus dem Speicher 0000, inkrementiert diesen Zähler in einer Schleife bis zum Erreichen von Null und springt dann zurück. Mit der Vorgabe des Zählers wird die Länge der Verzögerung programmiert.

```
0120  A2 9D    DLYA  LDX #$9D   ─────────────────────────────────┐
0122  A0 71    M2    LDY #$71   ───────────────────┐             │
0124  C8       M1    INY        ──┐ Innere         │ Äußere      │
0125  C0 00          CPY #$00     │ Schleife       │ Schleife    │
0127  D0 FB          BNE M1     ──┘                │             │
0129  E8             INX                           │             │
012A  E0 00          CPX #$00                      │             │
012C  D0 F4          BNE M2     ───────────────────┘             │
012E  E6 00          INC $00    Inkrementiere Verzögerungs-      │
                                zähler jedesmal, wenn äußere     │
                                Schleife durchlaufen ist         │
0130  A5 00          LDA $00                                     │
0132  C9 00          CMP #$00                                    │
0134  D0 EA          BNE DLYA   Schleife, bis Zähler = 0 ────────┘
0136  60             RTS
```

Bild 5.10: Verkehrssteuerung, Nacht (Programm 5.1)

Nachtsteuerung

Dies ist der einfachste Teil. In beiden Richtungen erzeugt das Programm gelbes Blinklicht. Man schaltet meist Ampeln an Kreuzungen mit geringem Verkehrsaufkommen auf derartiges Blinklicht (i. a. jedoch nur in der Richtung, die nicht vorfahrtberechtigt ist). Das entsprechende Flußdiagramm zu diesem Algorithmus zeigt Bild 5.9. Wie man sieht, sind die gelben Phasen für die beiden Fahrtrichtungen abwechselnd an. Sie werden beide jeweils für eine bestimmte Dauer ein- bzw. ausgeschaltet. Bild 5.10 zeigt das zu diesem Flußdiagramm gehörende Programm. Es besteht aus dem Hauptprogramm „NACHT" und einem Unterprogramm „DLYA". Wir wollen das Programm nun im Detail studieren.

Mit einem Blick auf Bild 5.7 stellen wir fest, daß das Datenrichtungsregister des VIA#1 6522 zuerst so geladen werden muß, daß die sechs niederwertigen Bits des IORA als Ausgabekanäle festgelegt werden, die dann die LED's treiben. Das DDRA hat die Speicheradresse A003 und das IORA die Adresse A001 (siehe Bild 3.6: 6522 Speicherbelegung).

Die beiden ersten Befehle laden das DDRA mit dem richtigen Wert:

```
NACHT      LDA  #$3F      „00111111"
           STA  $A003     DDRA VIA#1
```

Wir brauchen das IORA jetzt nur noch mit dem entsprechenden Byte zu laden, um die LED's wie gewünscht ein- oder auszuschalten. Die verschiedenen Bitmuster zur Adressierung der einzelnen LED-Paare finden Sie in Bild 5.11.

BINÄR	HEX	AMPEL
00000001	01	GRÜN A
00000010	02	GELB A
00000100	04	ROT A
00001000	08	GRÜN B
00010000	10	GELB B
00100000	20	ROT B

Bild 5.11: Bitmuster zur Adressierung der LED-Paare

Die nächsten beiden Programmzeilen schalten die gelbe Phase für die Richtung A ein, indem sie hexadezimal „02" in das IORA laden.

```
NACHT2     LDA  #$02      GELB A
           STA  $A001     SETZE IORA
```

Dann muß man eine Verzögerung veranlassen. Die Verzögerungskonstante wird in den Akkumulator geladen und dann in den Speicher 00 ab-

gelegt, von wo sie sich die Verzögerungsroutine DLYA wieder holen wird. Anschließend erfolgt der Unterprogrammaufruf für die Routine DLYA.

```
LDA  #$FC
STA  $00
JSR  DLYA
```

Wenn die eingestellte Verzögerungszeit abgelaufen ist, wird der hexadezimale Wert „10“ in das IORA geladen. Dadurch wird die gelbe Phase in der A-Richtung ausgeschaltet und gleichzeitig die Gelbphase in B-Richtung eingeschaltet. Wie vorhin wird dann die Verzögerungsdauer in den Speicher „00“ geladen und das Unterprogramm DLYA wieder aufgerufen:

```
LDA  #$10     GELB B
STA  $A001    IORA
LDA  #$FC
STA  $00
JSR  DLYA
```

Zum Schluß springt das Programm nach Ablauf der vorgegebenen Verzögerungszeit wieder zur Marke NACHT2, wo die Gelbphase A wieder ein- und die Gelbphase B wieder ausgeschaltet wird.

```
JMP  NACHT2
```

Die Funktionsweise des Programms sollte bis zu dieser Stelle klar sein. Sehen wir uns also das Unterprogramm DLYA an. Verzögerungsschleifen funktionieren so, daß ein Register oder Speicherplatz mit einem Startwert geladen und dann inkrementiert oder dekrementiert wird, bis ein vorgegebener Endwert erreicht ist. Da der Leser mit der Dekrementiermethode sicherlich schon gut vertraut ist (siehe „Programmierung des 6502“), wollen wir hier zur Abwechslung die Inkrementiermethode anwenden. Allerdings benötigt sie einige Befehle mehr. Einen Vorschlag zur Verbesserung des Programms geben wir am Ende dieses Abschnitts in einer Übungsaufgabe.

Bild 5.11 zeigt die Verzögerungsroutine DLYA. Da die gewünschte Verzögerungszeit in der Größenordnung von einigen Zehntelsekunden liegt, kann man sie nicht mit einer einzigen Schleife durchführen. Eine einzelne Schleife würde ein Register mit dem Wert 255 laden (hexadezimal FF) und dieses dann dekrementieren oder inkrementieren. Die erzielbare Verzögerung wäre nicht ausreichend. Um größere Verzögerungszeiten darzustellen, werden wir mehrere Schleifen ineinander verschachteln: eine innere Verzögerungsschleife und mindestens eine äußere Verzögerungsschleife, die immer dann ausgeführt wird, wenn die innere beendet ist. Sehen wir uns das Programm an. Das Register X wird als Zähler für

die äußere Schleife verwendet. Es wird mit dem hexadezimalen Wert 9D geladen. Warum wir gerade 9D wählen, wird später erklärt.

```
DLYA        LDX  #$9D
```

Der zweite Befehl in diesem Unterprogramm lädt das Y-Register mit dem hexadezimalen Wert 71. Y ist der Zähler der inneren Schleife:

```
M2          LDY  #$71
```

Die nächsten drei Instruktionen bilden zusammen die innere Schleife:

```
M1          INY
            CPY  #$00
            BNE  M1
```

Y wird solange inkrementiert, bis es den Wert 0 erreicht. Wenn die innere Zählschleife ausgezählt hat (also den Endwert 0 erreicht hat), wird der äußere Zähler X inkrementiert. Das erledigt die sechste Programminstruktion:

```
            INX
```

Wenn X inkrementiert ist, wird es mit Null verglichen, und solange der Endwert 0 noch nicht erreicht ist, erfolgt eine Verzweigung zurück zum Punkt M2 am Beginn der inneren Schleife:

```
            CPX  #$00
            BNE  M2
```

Die gesamte Verzögerung ist an diesem Punkt die Verzögerungszeit der inneren Schleife multipliziert mit der Anzahl der Durchläufe der äußeren Schleife.

Jedesmal, wenn diese äußere Verzögerungsschleife beendet ist, wird unser Hauptzähler im Speicher 00 um 1 erhöht:

```
            INC  $00
```

Dies ergibt eine dritte Schleife. Der Inhalt des Speichers 00 wird nach jedem Inkrementieren auf den Endwert 0 getestet. Wenn der Wert 0 erreicht ist, springen wir aus der Routine ins Hauptprogramm zurück. Solange der Wert 0 noch nicht erreicht ist, verzweigen wir zurück zum Punkt DLYA, also an den Anfang der allerersten Schleife und durchlaufen das ganze wieder.

```
            LDA  $00
            CMP  #$00
            BNE  DLYA
            RTS
```

Bild 5.12 zeigt die Gesamtstruktur des Programms mit seinen drei verschachtelten Schleifen und den Taktzykluszeiten der einzelnen Befehle. Die Gesamtverzögerung wird sein: Inhalt des Speichers 00 mal der Anzahl der Durchläufe der äußeren Schleife mal der Verzögerungszeit der inneren Schleife. Berechnen wir nun diese gesamte Verzögerung. Beginnen wir mit der inneren Schleife. Bei jedem Durchlaufen werden drei Befehle mit einer Gesamtdauer von sieben Taktzyklen ausgeführt. Der Einfachheit halber soll diese innere Schleife eine Verzögerung von einer Millisekunde ergeben. Die erste äußere Schleife wird dann dafür verantwortlich sein, daraus eine Verzögerung von 100 msec (= 0,1 sec) zu erzeugen.

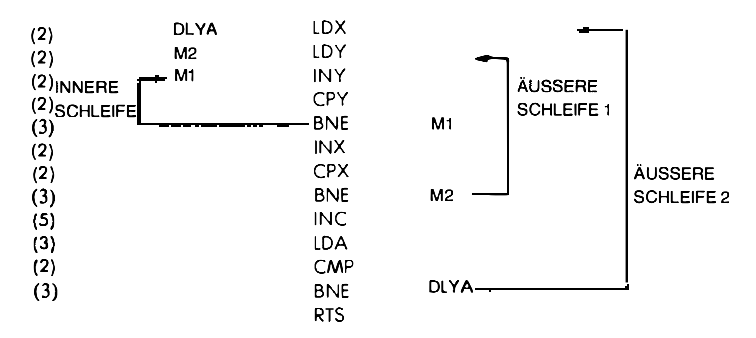


Bild 5.12: Zeitablauf der Routine DLYA

Beginnen wir mit dem Wert 80 (hexadezimal) im Register Y. Das ist dezimal 128 und genau die Mitte des Bereichs, den man mit 8 Bit darstellen kann. Von der inneren Schleife wird das Y-Register daher 128 mal inkrementiert. Die Gesamtdauer der Schleife wird also 7 mal 128 gleich 896 Mikrosekunden sein. Da wir für die innere Schleife eine Verzögerung von etwa 1000 Mikrosekunden erreichen wollen, müssen wir das Y-Register also mit einem anderen Wert laden. Diesen berechnen wir jetzt. Wir wollen diesen Wert N so wählen, daß N x7 = 1000 ist. N muß daher gleich 1000/7 = 142,86 sein. Die nächste ganze Zahl ist 143. Da wir in diesem speziellen Verzögerungsprogramm den Inhalt des Y-Registers nicht dekrementieren, sondern inkrementieren, werden wir das Y-Register mit dem Wert 256−143 = 113 (dezimal) oder 71 hexadezimal laden.

Weiter wollen wir die Dauer der durch die erste äußere Schleife erzeugten Verzögerung berechnen. Ein einmaliges Durchlaufen der äußeren Schleife wird in einer Verzögerung resultieren, die gleich der Dauer der ersten Programminstruktion (bei der Adresse DLYA) plus der Dauer der inneren Schleife plus der Dauer der drei folgenden Befehle bis einschließlich dem Sprungbefehl BNE M2 ist. Die gesamte Dauer ist also:

2 + 7x143 + 7 = 1010 Mikrosekunden.

Wir wollen erreichen, daß diese äußere Schleife eine Verzögerung von 0,1 sec oder 100·000 Mikrosekunden ergibt. Die Zahl der Schleifendurchläufe P muß also so gewählt werden, daß 1010 x P = 100·000 ist. Hieraus ergibt sich P zu P = 100·000/1010 = 99.

Da wir eine Inkrementiermethode verwenden, muß wiederum die in das X-Register geladene Zahl so gewählt werden, daß sie 99 mal inkrementiert wird, bevor sie den Endwert „00" erreicht. Die Zahl, die wir ins X-Register laden müssen, ist daher gleich 256−99 = 157 dezimal oder 9D hexadezimal. Vergewissern wir uns, daß wir die richtige Verzögerungszeit erhalten: die äußere Schleife braucht genau 99 x 1010 = 99·990 Mikrosekunden. Die restlichen vier Instruktionen am Ende der DLYA-Routine benötigen 5 + 3 + 2 + 3 = 13 µsec. Zwei µsec müssen wir für die erste Programminstruktion noch addieren.

Die endgültige Gesamtverzögerung für einen Durchlauf der DLYA-Routine ist also 99990 + 15 = 100·005 µsec. Diese Verzögerung ist fast genau gleich 0,1 sec. In der Tat ist das so nahe bei 0,1 sec, daß Sie die Verzögerung mit der Stoppuhr nachmessen und sich so überzeugen können, wie genau diese Methode funktioniert.

Am Schluß noch eine Warnung: denken Sie daran, daß dieses Unterprogramm eine *Inkrementier*technik verwendet. Die Zahl, die in den Speicher 00 gelegt wird, steuert die Dauer der Verzögerung in Einheiten von einer zehntel Sekunde. Allerdings muß diese Zahl im Speicher 00 das *Komplement* zu 256 der eigentlichen Anzahl von Zehntelsekunden sein, da sie solange *inkrementiert* wird, bis Null erreicht ist. Man muß also mit anderen Worten zur Erzeugung einer 0,4 sec Verzögerung nicht den Wert 4 in den Speicher 00 laden, sondern den Wert 256−4 = 252 dezimal bzw. FC hexadezimal. Genau das haben wir im Programm in Bild 5.10 (Nachtsteuerung) gemacht.

Es ist nun an der Zeit, diese Verzögerungsroutine in einigen Punkten zu verbessern:

Übungsaufgabe 5.1: *Schreiben Sie das Verzögerungs-Unterprogramm unter Verwendung einer Dekrementiertechnik anstelle der Inkrementiertechnik um. Berechnen Sie die Startwerte, mit denen die Register X und Y geladen werden müssen, neu, so daß die resultierende Verzögerung durch die Routine wieder etwa 0,1 sec ist. Welchen Vorteil bietet diese Dekrementiertechnik gegenüber einer Inkrementiertechnik?*

Vorsicht: Wenn Sie sich im Verzögerungsunterprogramm für eine Dekremmentiertechnik entscheiden, dürfen Sie nicht vergessen, die Zeilen 010A und 0116 im Hauptprogramm abzuändern. Vor dem ersten Aufruf der Routine muß eine andere Konstante geladen werden.

Übungsaufgabe 5.2: *Ändern Sie das Programm so, daß die Ampeln alle Sekunde blinken. Verkürzen Sie das Programm ferner durch Verwendung des EOR-Befehls zum Umschalten der Ampeln.*

Tagsteuerung

In diesem Betriebszustand durchläuft jede der Ampeln in der gewohnten Weise die Rot-, Grün- und Gelbphasen. Solange die Ampeln in der Richtung A grün oder gelb zeigen, zeigt die Ampel in der Richtung B Rot und umgekehrt. Das Flußdiagramm zu diesem Steueralgorithmus sehen Sie in Bild 5.13. Die Pfeile rechts neben dem Flußdiagramm sind ein Maß für die Zeitspanne, während der die jeweilige Phase angeschaltet ist. Wenn wir die Dauer der Grünphase in Richtung A D1 nennen, die der Gelbphase D2 und die der Grün- bzw. Gelbphase in der Richtung B D3 bzw. D4, dann ersehen wir aus dem Diagramm, daß die Gesamtdauer eines Zyklus D1 + D2 + D3 + D4 ist.

Bei richtigen Kreuzungen müssen diese Zeiten gewisse Randbedingungen erfüllen. Normalerweise liegt der Gesamtzyklus zwischen einer und zwei Minuten. Die obere Grenze ist dadurch gegeben, daß die meisten Autofahrer eine Rotphase von mehr als zwei Minuten in irgendeiner Fahrtrichtung nicht akzeptieren: sie fahren einfach bei Rot los, wenn ihre Geduld erschöpft ist, da sie den Schluß ziehen, die Ampel sei defekt. Weiter gibt es eine untere Grenze, die dadurch bedingt ist, daß Fahrzeuge oder Fußgänger eine gewisse Zeit benötigen, die Kreuzung zu räumen, wenn sie diese einmal befahren bzw. betreten haben. Die Dauer der Gelbphase nennen wir auch Freigabezeit. Das ist die Zeit, die ein Fahrzeug benötigt, um die Kreuzung zu räumen. Die Grünphase kann irgendeine minimale Dauer haben, solange keine Fußgänger die Kreuzung passieren. Sollen jedoch auch Fußgänger passieren können, muß die minimale Dauer der Rotphase so gewählt werden, daß ein Fußgänger die Straße sicher überqueren kann. So ist beispielsweise die Dauer der Rotphase in Richtung B gleich D1 + D2.

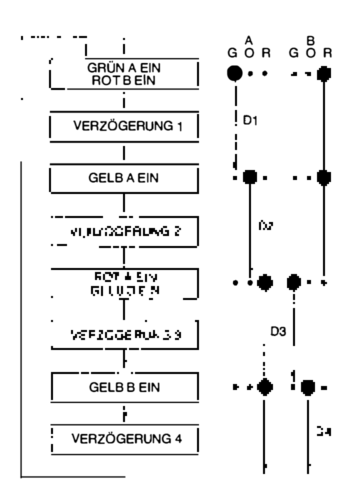


Bild 5.13: Tagsteuerung (Ausschaltbefehle nicht aufgeführt)

Wenn wir beispielsweise annehmen, daß die Mindestdauer für die Gelbphase in Richtung A gleich 3 sec ist, und daß die Mindestdauer der Rotphase in Richtung B 10 sec ist, dann können wir Bild 5.13 entnehmen, daß die Mindestdauer der Grünphase in Richtung A gleich D1 = 10 − 3 = 7 sec ist. Oder mathematisch:

Setzen wir:

GRÜN A = D1
GELB A = D2
GRÜN B = D3
GELB B = D4

Dann folgt:

ROT A = D3 + D4
ROT B = D1 + D2

Außerdem ist der Gesamtzyklus festgelegt und D1 + D2 + D3 + D4 = KONSTANT.

Wir werden in unserem Programm schnellere Zyklen als im tatsächlichen Straßenverkehr wählen. Das hat einfach den Grund, daß es frustrierend ist, eine oder zwei Minuten warten zu müssen, nur um die korrekte Funktion der Ampelanlage zu testen. Für unsere Belange ist eine Zykluszeit von etwa 10 bis 20 sec für Testläufe wünschenswert und der Leser sollte inzwischen auch schon über die nötigen Kenntnisse verfügen, die Verzögerungen anders einzustellen, so daß man den Mikrocomputer an eine richtige Ampelanlage anschließen könnte. Bild 5.14 zeigt das Programm.

```
0140  A9 3F       TAG    LDA  #$3F    „00111111“
0142  8D 03  A0          STA  $A003   Setze VIA#1 DDRA = $3F
                                      für Ausgabemodus
0145  A9 21       TAG1   LDA  #$21
0147  8D 01  A0          STA  $A001   ROTB, GRÜNA
014A  A9 D0              LDA  #$D0    Dezimal −48
014C  85 00              STA  $00     DLYA-Zähler in Speicher $0000
014E  20 20  01          JSR  DLYA    Verzögerungsroutine
0151  A9 22              LDA  #$22
0153  8D 01  A0          STA  $A001   GELB A, ROTB
0156  A9 EA              LDA  #$EA    Dezimal −22
0158  85 00              STA  $00     DLYA-Zähler
015A  20 20  01          JSR  DLYA    Verzögerungsroutine
015D  A9 0C              LDA  #$0C
015F  8D 01  A0          STA  $A001   ROTA, GRÜNB
0162  A9 D0              LDA  #$D0    Dezimal −48
0164  85 00              STA  $00     DLYA-Zähler
```

Bild 5.14: Ampelsteuerung, Tag (Programm 5.2) (Steckerleiste A mit Stecker H1 verbinden)

```
0166  20  20  01      JSR   DLYA    Verzögerungsroutine
0169  A9  14          LDA   #$14
016B  8D  01  A0      STA   $A001   ROTA, GELB B
016E  A9  E8          LDA   #$E8    Dezimal -24
0170  85  00          STA   $00     DLYA-Zähler
0172  20  20  01      JSR   DLYA    Verzögerungsroutine
0175  4C  45  01      JMP   TAG1    Von vorne
```

Bild 5.14: Ampelsteuerung, Tag (Programm 5.2) (Fortsetzung)

Wie im vorhergehenden Programm muß im Datenrichtungsregister DDRA zur Steuerung der 6 LED's die Ausgabefunktion festgelegt werden. Das machen wir mit den beiden ersten Programminstruktionen:

```
TAG     LDA  #$3F     „00111111“
        STA  $A003    DDRA
```

Dann werden mit den beiden nächsten Befehlen die Richtung A auf Grün und die Richtung B auf Rot geschaltet. Dazu wird das passende Bitmuster (21 hexadezimal) in das E/A-Register IORA geladen:

```
TAG1    LDA  #$21
        STA  $A001
```

Dann wird die Dauer der nachfolgenden Verzögerung festgelegt, indem die Speicheradresse 00 mit einem bestimmten Wert geladen wird. Dann wird die Verzögerungsroutine aufgerufen:

```
        LDA  #$D0
        STA  $00
        JSR  DLYA
```

Dann wird der Vorgang für die Gelbphase in Richtung A, für die Rotphase in Richtung A, für die Grünphase in Richtung B und schließlich für die Gelbphase in Richtung B wiederholt, bevor das Programm an den Anfang zurückspringt:

```
        LDA  #$22     GELB A UND ROT B
        STA  $A001
        LDA  #$EA     VERZÖGERUNGS-
                      KONSTANTE
                      -22 DEZIMAL
        STA  $00
        JSR  DLYA     VERZÖGERUNGSROUTINE
        LDA  #$0C     ROT A UND GRÜN B
        STA  $A001
        LDA  #$D0     VERZÖGERUNGS-
                      KONSTANTE
                      -64 DEZIMAL
```

```
STA  $00
JSR  DLYA     VERZÖGERUNGSROUTINE
LDA  #$14     ROT A UND GELB B
STA  $A001
LDA  #$E8     VERZÖGERUNGS-
              KONSTANTE
              -24 DEZIMAL
STA  $00
JSR  DLYA     VERZÖGERUNGSROUTINE
JMP  TAG1     VON VORNE
```

Der Leser sollte sich davon überzeugen, daß das Programm mit dem Flußdiagramm in Bild 5.13 genau übereinstimmt. Es sollten dabei keine Verständnisschwierigkeiten auftreten. Es sei dem Leser dringend empfohlen, auch andere Zeitkonstanten auszuprobieren und sich zu überzeugen, daß der Zeitablauf so ist, wie er es erwartet. Wir wollen nun einige Verbesserungen des Verkehrssteuerprogramms betrachten.

Beispielsweise kann man das Programm so ändern, daß die Gelb- und Rotphase und die Zykluszeit durch diejenige Zeit bestimmt werden, während der einer der Schalter nach dem Starten des Programms gedrückt wird.

Übungsaufgabe 5.3: *Erstellen Sie einen „dynamischen Algorithmus“: Die Dauer der Grünphase in Fahrtrichtung A wird jedesmal, wenn von der Induktionsschleife (einem Schalter) ein Impuls registriert wird, um fünf Sekunden bis zu einer Maximaldauer von drei Minuten ausgedehnt.*

Übungsaufgabe 5.4: *Lassen Sie durch Verwendung der Schalter auch Grünanforderungen von Fußgängern zu. Fußgänger sollten so schnell wie möglich Grün bekommen, wobei aber die Mindestphasendauer berücksichtigt werden muß.*

Übungsaufgabe 5.5: *Installieren Sie einen „Sonderschalter für die Polizei“: Wenn einer der Schalter gedrückt wird, sollen die Ampelphasen manuell durchlaufen werden. Wird der Schalter zweimal kurz hintereinander gedrückt, soll die Steuerung auf Automatik zurückschalten.*

Punktmatrix LED

Wir werden eine 5 x 7 Punktmatrix aus LED’s nach Bild 5.15 als Anzeige verwenden. Dieser Matrixtyp wird in zahlreichen Anwendungen eingesetzt. So verwenden beispielsweise Punktmatrixdrucker oft eine 5 x 7 Matrix zur Ausgabe von Zeichen auf Papier. Fernsehmonitore oder CRT-Displays benutzen ebenfalls die Punktmatrix zur Darstellung von Zeichen

auf dem Bildschirm. 5 x 7 ist die Standard-Mindestpunktmatrix für eine annehmbare Zeichendarstellung. Im Hinblick auf eine gute Lesbarkeit ist sie aber nicht das Optimum. Größere Punktmatrizen, wie etwa die 7 x 9 Matrix, werden für verbesserte Lesbarkeit bei allerdings höheren Kosten eingesetzt. In diesem Beispiel werden wir eine 5 x 7 LED-Matrix direkt

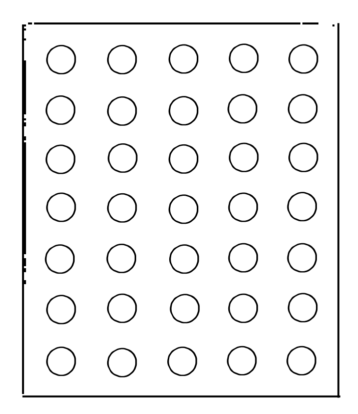


Bild 5.15: eine 5 x 7 Punktmatrix aus LED's

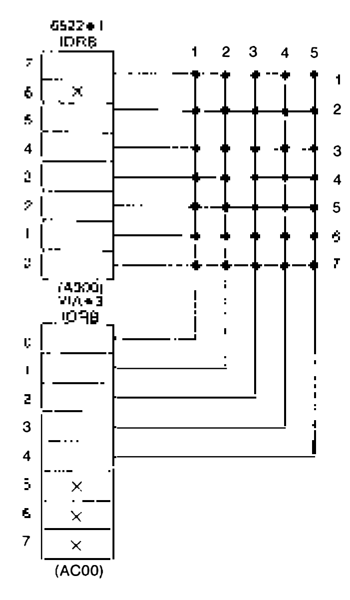


Bild 5.16: Anschluß der 5 x 7 LED-Matrix

an die E/A-Register B des 6522 #1 und des 6522 #3 anschließen. Eigentlich sollten die LED's über Treiber betrieben werden, um eine ausreichende Lichtausbeute zu erreichen. Um die Zahl der verwendeten Bauteile klein zu halten, werden wir die LED's jedoch direkt anschließen. Das bedeutet natürlich, daß die LED's auf der Platine nur schwach leuchten und die Anzeige unter Umständen etwas schwer zu erkennen ist. Eine Verbesserung erzielen Sie leicht durch Vorschalten von Treibern. Wie man die Punktmatrix anschließt, zeigt Bild 5.16. Die sieben Zeilen, die von 1 bis 7 durchnumeriert sind, werden an die Bits 7,5,4,3,2,1 und 0 des E/A-Registers B (IORB) des 6522#1 angeschlossen. Bit 6 dieses IORB's ist beim SYM nicht verfügbar, da der Monitor das Bit 6 bei der Eingabe über Kassette benötigt. Der Status von Bit 6 ist daher bei den folgenden Betrachtungen unwichtig.

Die fünf Spalten der LED-Anzeige, die wir von 1 bis 5 durchnumerieren, sind mit den Bits 0 bis 4 des IORB des 6522#3 verbunden. Man sieht das gut in Bild 5.16. Die beiden IORB's haben die Speicheradressen A000 und AC00.

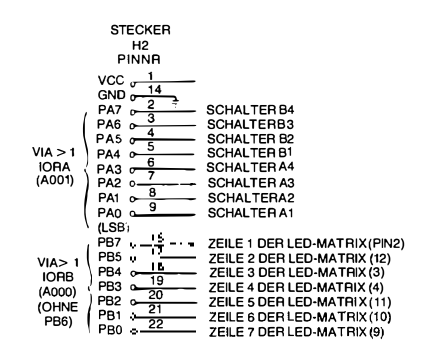


Bild 5.17a: Anschluß der LED's (Zeilen)

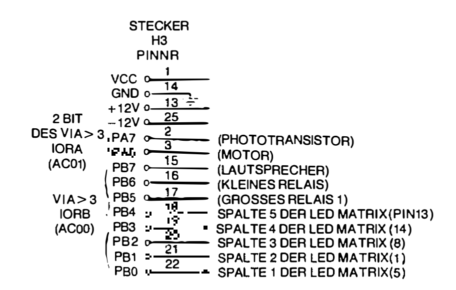


Bild 5.17b: Anschluß der LED's (Spalten)

Das Hauptproblem ist die Wahl einer geeigneten Kombination aus Zeilen und Spalten zur Anzeige eines Zeichens in Form einer Punktmatrix. Jeder Buchstabe des Alphabets kann mit einer 5 x 7 Matrix dargestellt werden. Als Beispiel werden wir hier alle Hexadezimalzeichen darstellen, d. h. die Ziffern von 0 bis 9 und die Buchstaben von A bis F. Wir wollen die Kodierung jetzt studieren.

Eine LED der Matrix im Zustand „an" wird durch ein Bit „0" repräsentiert. Eine ausgeschaltete LED wird durch eine „1" repräsentiert. Man trifft die Wahl deshalb so, weil eine LED dadurch eingeschaltet wird, daß man ihre Spalte auf Null legt. Das Muster zur Darstellung einer „0" zeigt Bild 5.18. Natürlich kann der Benutzer jedes andere Muster oder andere Kodierungen frei wählen. Als eine Übung könnte der Benutzer beispielsweise die „0" mit rechtwinkligen Ecken statt mit runden darstellen wollen. Es sollte einfach sein, die Tabelle entsprechend abzuändern.

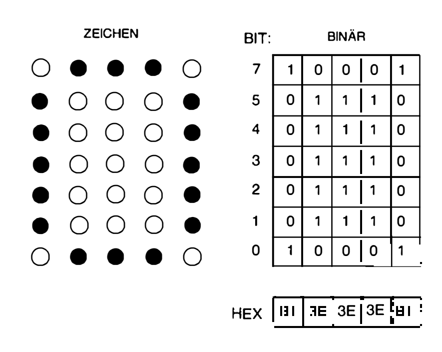


Bild 5.18: Anzeige einer „0"

ZEICHEN

BIT:	BINÄR				
7	1	1	0	1	1
5	1	1	0	1	1
4	1	1	0	1	1
3	1	1	0	1	1
2	1	1	0	1	1
1	1	1	0	1	1
0	1	1	0	1	1
HEX:	BF FF	BF FF	00 40	BF FF	BF FF

Bild 5.19: Anzeige einer „1"

Auf der rechten Seite von Bild 5.18 steht das binäre Äquivalent des kodierten Zeichens. Unter der binären Tabelle finden Sie die hexadezimale Darstellung. Der Leser sollte daran denken, daß Zeile 6 nicht belegt ist. Ihr Wert spielt keine Rolle, kann also beliebig als „0“ oder „1“ angenommen werden. Sehen wir uns zum Beispiel den hexadezimalen Kode für das Zeichen „0“ in Bild 5.18 an. Die erste Spalte hat den Wert „1000001“, oder genauer „1x000001“, wobei das „x“ den Wert von Bit 6 darstellen soll, das nicht benutzt wird. Nehmen wir als Beispiel an, Bit 6 habe den Wert 0. Der Wert der ersten Spalte ist dann also „10000001“ oder „81“ hexadezimal.

Ganz ähnlich ist der Wert für die zweite Spalte (Bit 6 wird als 0 angenommen) gleich „00111110“ oder „3E“ hexadezimal.

Die fünf Spalten für die Zahl Null sind daher:

81, 3E, 3E, 3E, 81.

Sehen wir uns nun das Zeichen „1“ an. Bild 5.19 zeigt das Zeichen und rechts daneben die binäre Kodierung.

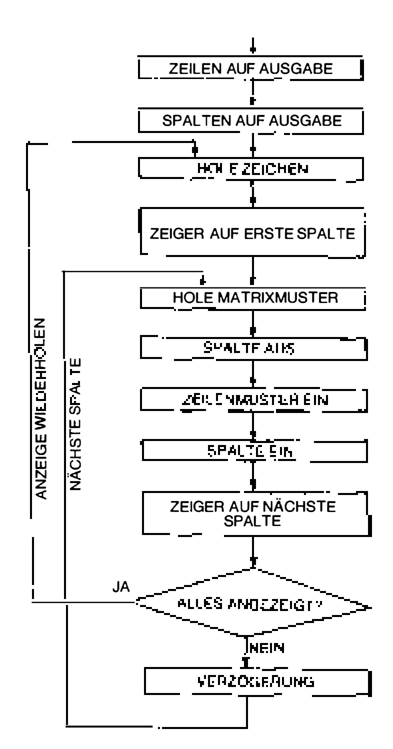


Bild 5.20: Treiberprogramm für eine Punktmatrix aus LED’s

Unter der Annahme, daß Bit 6 gleich „0" gewählt wird, erhält man die hexadezimale Darstellung:

BF, BF, 00, BF, BF

Wählen wir für Bit 6 eine „1", so erhalten wir:

FF, FF, 40, FF, FF

Solange wir Bit 6 nicht für irgendeinen anderen Zweck definieren, können wir es in jeder Zeile beliebig zu „0" oder „1" wählen.

Eine vollständige Tabelle der kodierten Zeichen von „0" bis „F" finden Sie in Bild 5.21.

Übungsaufgabe 5.7: *Zeichnen Sie unter Verwendung dieser Tabelle alle Zeichen von „0" bis „F".*

Übungsaufgabe 5.8: *Schreiben Sie die Tabelle einheitlich neu, indem Sie Bit 6 stets „0" wählen.*

Das Flußdiagramm für das LED-Punktmatrixprogramm sehen Sie in Bild 5.21. Für die Zeilen und Spalten wird die Ausgabefunktion festgelegt. Dazu werden die Datenrichtungsregister des benutzten 6522 entsprechend geladen. Dann muß das Punktmuster des Zeichens angezeigt werden. Die Punkte werden spaltenweise angesteuert. Für jedes Zeichen muß das Programm daher auf fünf hintereinander in der Punktmatrix-Tabelle stehende Eintragungen zugreifen. Sie entsprechen den fünf Spalten, die zur Anzeige des Zeichens gebraucht werden. Das Programm ist als Endlosschleife ausgebildet und zeigt das Zeichen dauernd an. Das Punktmuster wird angezeigt, indem die Spalten ausgeschaltet werden (Löschen des vorhergehenden Zeichens), dann wird das Punktmuster der gewünschten Spalte geladen und die Spalte, auf der es angezeigt werden soll, eingeschaltet. Anschließend muß die nächste Spalte angezeigt werden. Damit dem Betrachter alle Punkte in gleicher Helligkeit erscheinen, müssen sie für die selbe Zeit eingeschaltet werden. Weiter müssen alle Spalten in weniger als einer Zehntelsekunde durchlaufen werden, damit die Anzeige flimmerfrei ist. Die Verzögerungsroutine am Ende des Programms wurde dementsprechnd gewählt. Das Programm finden Sie auf der nächsten Seite.

Zeichen	TAB. Adr.	SP1	SP2	SP3	SP4	SP5
0	90	81	3E	3E	3E	81
1	95	FF	FF	00	FF	FF
2	9A	DE	7C	7A	76	CE
3	9F	DD	76	76	76	C9
4	A4	F3	EB	DB	00	FB
5	A9	05	76	76	76	79
6	AE	C1	76	76	76	D9
7	B3	7F	7F	7F	7F	00
8	B8	C9	76	76	76	C9
9	BD	CD	76	76	76	C1
A	C2	E0	DB	7B	DB	E0
B	C7	00	76	76	76	C9
C	CC	C1	7E	7E	7E	DD
D	D1	00	7E	7E	7E	C1
E	D6	00	76	76	76	76
F	DB	00	77	77	77	77

Die Tabelle liegt im Speicherbereich von 0090 bis 00DF.

Bild 5.21: Die Tabelle für die Punktmatrix

Verbindung: Steckerleiste A mit Stecker H2 verbinden und Steckerleiste AA mit Stecker H3 verbinden

Das Programm holt die Tabellen-Adresse (die 8 niederwertigen Bits der Adresse, die 8 höherwertigen sind Null) des Zeichens aus der Speicheradresse 0001, springt dann zur Tabelle, die in Bild 5.21 gezeigt ist, holt von dort das Bytemuster des gewählten Zeichens und stellt dieses dann auf der LED-Matrix dar.

Laden Sie vor Ausführung des Programms diese Tabellen-Adresse des gewünschten Zeichens in den Speicher 0001.

Die Bytemuster aller Zeichen sollten wie in Bild 5.21 gezeigt auf der nullten Seite abgelegt sein.

(die 8 höherwertigen Bits sind auf der Seite Null definitionsgemäß gleich Null)

Anmerkung: 1) Ein Zeichengenerator kann diese Tabelle ersetzen.

2) Die verwendete Matrix ist eine 5x7-Matrix, d. h. zur Darstellung des Zeichens sind 7 Bit pro Spalte nötig, unsere Tabelle gibt aber 8 Bits an; das liegt daran, daß das Programm das E/A-Register B des VIA#1 zur Ansteuerung der 7 Zeilen verwendet, und nur 7 Bits dieses Registers verfügbar sind. Bit 6 ist für die Funktion ON BOARD CASSETTE IN vorgesehen und daher nicht verfügbar.

```
0180  A9 BF       MATRIX LDA  #$BF    Vor Ausführung sollte 0001
                                      geladen sein
0182  8D 02 A0           STA  $A002   definiert DDRB VIA#1 = BF
                                      zur Steuerung der 7 Zeilen
0185  A9 1F              LDA  #$1F
```

Bild 5.22: Standard-Matrix-Anzeige mit Led's (Programm 5.3)

```
0187  8D 02 AC          STA  $AC02     definiertDDRB VIA#3 = 1F
                                       zur Steuerung der 5 Spalten
018A  A9 00             LDA  #$00
018C  85 03             STA  $03       setzt MSB-Adresse des Zeichens
                                       = 00 im Speicher 0003
018E  A2 00             LDX  #$00
0190  A5 01    WDHZEI   LDA  $01       überträgt 8LSB-Adresse des
                                       Zeichens von 0001 nach 0002
0192  85 02             STA  $02
0194  A0 10             LDY  #$10      setzt (Y) = $10 zum Löschen
                                       der letzten Spalte
0196  A1 02    NXTSPA   LDA  ($02,X)   lädt aktuelles Byte des
                                       Zeichens in A
0198  8E 00 AC          STX  $AC00     sperrt alle Spalten vor
                                       Ansteuern der Zeile
019B  8D 00 A0          STA  $A000     Ansteuerung der Zeile
019E  8C 00 AC          STY  $AC00     Einschalten aktueller Spalte
01A1  E6 02             INC  $02       erhöht Adresse in 0002
                                       für nächste Spalte
01A3  98                TYA
01A4  4A                LSR  A         schiebt (Y) um 1 Bit nach rechts
                                       zum Einschalten der nächsten
                                       Spalte
01A5  A8                TAY
01A6  C0 00             CPY  #$00      (Y) = 00 bedeutet: alle 5 Spalten
                                       sind durchlaufen
01A8  D0 03             BNE  DLY3      falls noch nicht, springe nach
                                       DLY3 in Verzögerungsroutine (1).
                                       Falls ja, wiederhole Anzeige des
                                       gesamten Zeichens
01AA  4C 90 01          JMP  WDHZEI    wiederhole Anzeige
01AD  A2 7F    DLY3     LDX  #$7F      Zähler = 7F
01AF  E8       Schleife INX
01B0  E0 00             CPX  #$00
01B2  30 FB             BMI  Schleife  Schleife, bis (X) = 0
01B4  4C 98 01          JMP  NXTSPA    bei (X) = 0 nächste Spalte
```

Anmerkungen: 1) Die Verzögerung ist deswegen nötig, weil ohne sie die letzte Spalte länger eingeschaltet wäre und damit heller erschiene als die restlichen 4 Spalten.

2) Die hier vorgestellte Kompensationsmethode löst das Problem nur teilweise. Die Helligkeiten sind immer noch nicht gleich, da i. a. in jeder Spalte eine verschiedene Anzahl von LED's leuchtet. Um auch das zu lösen, müßte ein komplizierteres Programm geschrieben werden, das bei dieser Kompensation die Zahl der pro Spalte eingeschalteten LED's mit in Betracht zieht.

Bild 5.22: (Fortsetzung)

Sehen wir uns das Programm wieder im Detail an. Die ersten vier Befehle des Programms legen den Inhalt der Datenrichtungsregister für die Zeilen und Spalten fest und definieren die Ausgabefunktion:

```
MATRIX     LDA  #$BF
           STA  $A002     VIA #1 = 7 ZEILEN
           LDA  #$1F
           STA  $AC02     VIA #3 = 5 SPALTEN
```

In diesem Programm ist vereinbart, daß die Tabellenadresse des darzustellenden Zeichens auf der Seite 0 im Speicher „01" liegt. Die Adresse des darzustellenden Zeichens selbst sei beispielsweise gleich 90 für das Zeichen „0", 95 für das Zeichen „1" usw., wie in Bild 5.21 gezeigt. (Eine verbesserte Version des Programms wird weiter unten vorgeschlagen.) Wollen wir zum Beispiel das Zeichen „2" darstellen, dann muß in den Speicher 01 der Wert 9A geladen worden sein. Da wir in der Tabelle auf fünf aufeinanderfolgende Eintragungen für die fünf Spalten des darzustellenden Zeichens zugreifen müssen, müssen wir nacheinander die Adressen 9A,9B,9C,9D und 9E erzeugen. Um den ursprünglichen Zeiger „9A" des Zeichens nicht zu zerstören, werden wir zusätzlich die beiden Speicher 02 und 03 verwenden, die den aktuellen Zeiger auf die gerade dargestellte Spalte enthalten. Da wir uns auf der Seite Null befinden, wird der Inhalt des Speichers 03 auf Null gesetzt (höherwertiges Byte der Adresse). Das führen die beiden nächsten Befehle aus:

```
           LDA  #$00
           STA  $03
```

Wir wollen annehmen, daß das X-Register bei jedem Eintritt in die Haupt-Anzeigeschleife den Wert „00" besitzt. Wir werden es zum Löschen der Ausgaberegister heranziehen:

```
           LDX  #$00
```

Die erste Spalte, auf die wir zeigen werden, ist die, deren Adresse im Speicher 01 steht (Speicher 01 enthält den Zeiger auf die Einsprungadresse in der Zeichentabelle). Daher übertragen wir den Inhalt des Speichers 01 nach 02:

```
WDHZEI     LDA  $01
           STA  $02
```

Das Y-Register wird als Spaltenzähler und gleichzeitig zum Einschalten der jeweiligen Spalte eingesetzt. Es wird mit dem Wert „10" initialisiert, um auf die erste Spalte zuzugreifen:

```
           LDY  $10
```

Die „1“ wird dann um ein Bit nach rechts durchgeschoben, um auf die nächste Spalte zuzugreifen, etc. Wenn die Eins ganz durch das Register durchgeschoben ist, sind alle 5 Spalten des Zeichens angezeigt und die Schleife wird wieder von vorne begonnen. Da das Register nicht nur zum Einschalten der jeweiligen Spalte, sondern auch zum Hochzählen bis fünf benutzt wird, nennen wir es Schiebe-Zähler. Das Matrixmuster der aktuellen Spalte gewinnen wir, indem wir die Eintragung aus der Tabelle laden, deren Adresse in 02 steht:

```
NXTSPA      LDA  ($02,X)
```

Das aktuelle Matrixmuster steht nun im Akkumulator. Dieses wollen wir nun anzeigen. Alle Spalten werden zuerst durch Laden einer Null in das IORB gesperrt:

```
            STX  $AC00
```

Der Akkumulatorinhalt wird dann über das IORB ausgegeben, um die Zeilen anzusteuern:

```
            STA  $A000
```

Zum Schluß muß die aktuelle Spalte eingeschaltet werden und die angewählten LED's werden aufleuchten:

```
            STY  $AC00
```

Eine LED wird nur dann aufleuchten, wenn sie an die aktivierte Spalte und an eine geerdete (0) Zeile angeschlossen ist. Jede 0 im Punktmuster wird den entsprechenden Punkt in der angewählten Spalte zum Leuchten bringen.

Der Inhalt des Speichers „02“ wird anschließend inkrementiert, damit die Adresse auf die nächste Matrixspalte in der Zeichentabelle zeigt. Wir müssen dann unseren Spaltenzähler um ein Bit nach rechts verschieben und nachsehen, ob wir schon alle Spalten angezeigt haben oder nicht:

```
            INC  $02
            TYA              Das Y-Register kann nicht direkt
                             nach rechts geschoben werden
            LSR  A
            TAY              Resultat zurück nach A
            CPY  #$00
            BNE  DLY3
            JMP  WDHZEI
```

Da es nicht möglich ist, das Y-Register direkt um ein Bit nach rechts zu verschieben, muß es zuerst in den Akkumulator gebracht, dieser dann nach rechts geschoben und der Akkumulatorinhalt wieder in das Y-Regi-

ster zurückgebracht werden. Der Inhalt des Y-Registers wird dann auf den Wert „0“ getestet (an dieser Stelle läßt sich leicht eine Verbesserung des Programms durchführen). Hat das Y-Register den Wert „0“, so sind wir fertig und haben alle 5 Spalten angezeigt. Andernfalls müssen wir nun eine Verzögerung einführen, während der die LED’s eingeschaltet bleiben, und dann zur nächsten Spalte übergehen:

```
DLY3        LDX  #$7F
SCHLEIFE    INX
            CPX  #$00
            BMI  SCHLEIFE
            JMP  NXTSPA
```

Das X-Register wird hier als Zähler verwendet. Indem das X-Register mehrfach hintereinander inkrementiert wird, erhält man die inzwischen wohlbekannte Verzögerung. Anschließend erfolgt der Rücksprung zur Adresse NXTSPA.

Programmverbesserungen: Wir wollen das Programm durch Verminderung der Anzahl der benötigten Befehle verbessern. Hierzu betrachten wir zuerst Änderungen einzelner Befehle, anschließend werden wir Verbesserungen der Programmfunktionen studieren.

Übungsaufgabe 5.9: *Schreiben Sie die DLY3-Routine so um, daß sie weniger Befehle benötigt.*

Übungsaufgabe 5.10: *Betrachten Sie die drei letzten Befehle der Routine NXTSPA ab Adresse 01A6 in Bild 5.22. Können Sie einen anderen Weg angeben, wie man testen kann, ob das letzte „1“-Bit aus dem Y-Register herausgeschoben ist?*

Übungsaufgabe 5.11: *Schreiben Sie für das Programm eine zusätzliche Routine, so daß man, anstatt einen Zeiger auf die Zeichentabelle in den Speicher 01 laden zu müssen, lediglich den tatsächlichen Wert des anzuzeigenden Zeichens laden muß. Mit diesem Programm soll der Benutzer in der Lage sein, einen Wert zwischen „0“ und „F“ in den Speicher 01 zu laden und diesen durch das Programm richtig anzeigen zu lassen. Um das zu erreichen, muß man den wahren Wert in den Tabellenwert umrechnen. Das zeichen „0“ beispielsweise entspricht der Adresse „90“ (siehe Bild 5.21), eine „1“ entspricht „95“, usw. Die Gleichung für die Umrechnung ist: gesuchte Adresse = 90 + (Kode x 5).*

Anmerkung: Anstatt eine formale Multiplikation mit 5 durchzuführen, kann man eine Abkürzung wählen: Erinnern Sie sich, daß das Durchschieben nach links um ein Bit einer Multiplikation mit 2 entspricht, und das gilt: 5 = 2 + 2 + 1. Eine Multiplikation mit 4 kann man durch zweimaliges Schieben nach links erreichen.

Übungsaufgabe 5.12: *Schreiben Sie eine zusätzliche Routine, die eine Zeichenkette anzeigt. Nehmen Sie an, daß die Startadresse der Zeichenkette im Speicher 01 steht. Jedes Zeichen soll für eine Sekunde angezeigt werden. Die Zeichenkette wird durch irgendeinen Kode, der nicht zwischen 0 und F liegt, abgeschlossen. Das Programm soll nach Erreichen des Endes der Zeichenkette eine Pause von zwei Sekunden machen, und dann von vorne beginnen.*

Wir wollen nun Verbesserungen der Programmfunktionen studieren. Wir werden vier Schalter anschließen und ein Programm entwickeln, das den hexadezimalen Wert der aktuellen Schaltstellung anzeigt.

Anzeige von Schalterstellungen

Wir werden die Werte von vier Schaltern binär eingeben und das entsprechende hexadezimale Zeichen auf unserer LED-Matrix anzeigen. Das Flußdiagramm für diesen Algorithmus sehen Sie in Bild 5.23. Das Programm liest die Stellungen der vier Schalter, setzt den Zeiger (wie im vorhergehenden Programm erläutert) auf den Anfang der Umwandlungstabelle und berechnet dann die Adresse innerhalb der Tabelle für das anzuzeigende Zeichen. Man erhält die Adresse in der Tabelle für den binären Kode des aktuellen Zeichens durch Multiplikation des Wertes des Zeichens mit 5. Man kann sich das klarmachen, wenn man Bild 5.21 studiert. Die Adresse der ersten anzuzeigenden Spalte wird dann ausgerechnet und in den Speicher 01 auf der Seite 0 gelegt. Das letzte Programm wird dann zur Anzeige des Zeichens auf der LED-Matrix verwendet. Hier das Programm:

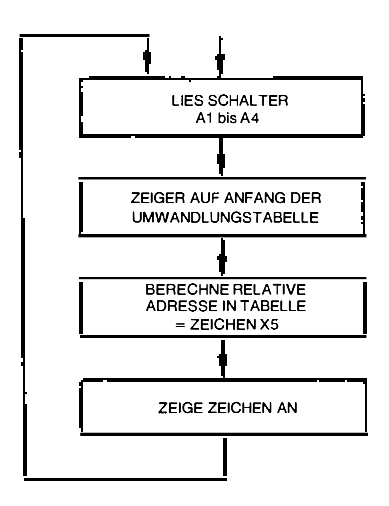


Bild 5.23: Anzeige von Schalterstellungen

Verbindung: Steckerleiste A mit Stecker H2 verbinden und Steckerleiste AA mit Stecker H3 verbinden

Das Programm liest die Stellungen der Schalter A1 bis A4 ein, berechnet den zugehörigen hexadezimalen Wert und zeigt ihn an.

Das Programm ruft das Programm 5.3 als Unterprogramm auf. Vor der Inbetriebnahme ändern Sie Programm 5.3 wie folgt ab:

1) Ersetzen Sie in Zeile 01AA das Byte 4C durch 60 (60 ist der Maschinenkode für RTS).
2) Ändern Sie die Zeitkonstante im Speicher 01AE in F0 um, da dieses Programm die letzte Spalte länger eingeschaltet läßt, als Programm 5.3.

```
0200  A9 00        RDCHA  LDA  #$00
0202  8D 03  A0           STA  $A003    Setze DDRA des VIA #1
                                        = 00 für Eingabemodus
0205  AD 01  A0           LDA  $A001    Lies Schalter B1–B4 und A1–A4
0208  29  0F              AND  #$0F     Unterdrücke B1–B4
020A  A8                  TAY           A1-A4 nach Y
020B  A2  90              LDX  #$90     Startadresse 90 der Tabelle
                                        über X nach 0001
020D  86  01              STX  $01
020F  A2  00              LDX  #$00     Additionszähler
0211  18           ADD    CLC           A enthält Schaltstellungen
0212  65  01              ADC  $01      Durchläuft Addition 5 mal
0214  85  01              STA  $01      90 + 5 x (A)
0216  98                  TYA
0217  E8                  INX           Schaltstellungen nach A
0218  E0  05              CPX  #$05     bei X=5 Addition fertig
021A  30  F5              BMI  ADD
021C  20  80  01          JSR  MATRIX   Unterprogramm MATRIX
                                        für Anzeige
021F  4C  00  02          JMP  RDCHA    von vorne mit neuer Schaltstellung
```

Bild 5.24: Verbesserte Matrixanzeige mit LED's (Programm 5.4)

Bild 5.24 zeigt das Programm. Die ersten beiden Anweisungen legen Tor A als Eingabetor fest, so daß die Schalter abgefragt werden können:

```
RDCHA      LDA  #$00
           A    $A003
```

Dann werden die Schalterstellungen A1 bis A4 eingelesen. Die Werte der Schalter B1 bis B4 werden vom Programm unterdrückt.

```
           LDA  $A001
           AND  #$0F        MASKIERE B1–B4
```

Der Inhalt von A, der nun die Stellung der vier Schalter angibt, wird dann in das Y-Register gerettet:

```
        TAY
```

Die Anfangsadresse der Tabelle ($90) wird dann in den Speicher 01 geladen:

```
        LDX  #$90
        STX  $01
```

Zu dieser Anfangsadresse werden wir die entsprechende Differenz hinzuaddieren, so daß wir auf die erste Spalte der Matrix des durch die Schalter festgelegten Zeichens Zugriff haben. Diese Differenz berechnet man durch Multiplikation des durch die Schalter eingestellten Wertes mit 5. Das Indexregister X wird als Zähler von 0 bis 5 eingesetzt. Es wird mit dem Wert 0 initialisiert:

```
        LDX  #$00
```

Sodann wird der Inhalt des Speichers 01 um den Inhalt des Akkumulators, der die eingelesene Schalterstellung enthält, vergrößert:

```
ADD     CLC
        ADC  $01
        STA  $01
```

Die CLC-Instruktion (clear carry, d. h. lösche das Übertragsflag) muß generell vor jede Addition gesetzt werden. Außerdem wollen wir annehmen, daß der Prozessor sich im dualen Modus befindet (der 6502 kann ja sowohl im dualen als auch im dezimalen Modus rechnen). Falls nicht anders angegeben, wird die CPU im allgemeinen im dualen Modus sein, da jede Reset-Operation alle Flags zurücksetzt und dadurch der duale Modus gesetzt ist.

Die eingelesene Schalterstellung wird dann aus dem Y-Register, wo sie zwischengespeichert war, in den Akkumulator zurückgeholt. Der Additionszähler X wird um 1 erhöht und auf der Wert 5 getestet:

```
        TYA
        INX
        CPX  #$05
        BMI  ADD
```

Solange der Wert 5 noch nicht erreicht ist, wird die Addition wiederholt. Sobald der Wert 5 erreicht ist, enthält der Speicher 01 die richtige Adres-

se und das Unterprogramm MATRIX (also das vorhergehende LED-Matrix-Programm) wird aufgerufen:

JSR MATRIX

Das Programm kehrt dann an den Anfang zurück, um den Schalter wiederum abzufragen und das durch diese gegebene Zeichen anzuzeigen:

JMP RDCHA

Tonerzeugung

Wir haben im letzten Kapitel gelernt, wie man einen Ton erzeugen kann, indem man einfach ein Rechtecksignal der gewünschten Frequenz an den Lautsprecher legt. Die Rechteckform des Ausgangssignals erreicht man ganz einfach durch abwechselndes Ein- und Ausschalten des Lautsprechers. Die Zeitspanne, während der der Lautsprecher ein- bzw. ausgeschaltet ist, nennen wir halbe Periode. Die Verzögerungsmessung kann man entweder mit Software oder aber mit Hardware durchführen, indem man den eingebauten Intervallzeitgeber des 6522 verwendet. Der eingebaute 6522-Zeitgeber wurde bereits in einem früheren Beispiel verwendet, also wenden wir diesmal eine *Software*-Technik zur Steuerung der Verzögerungsdauer an. Wir werden zuerst ein einfaches Programm zur Tonerzeugung entwickeln und dieses dann für die Erzeugung von Computermusik erweitern.

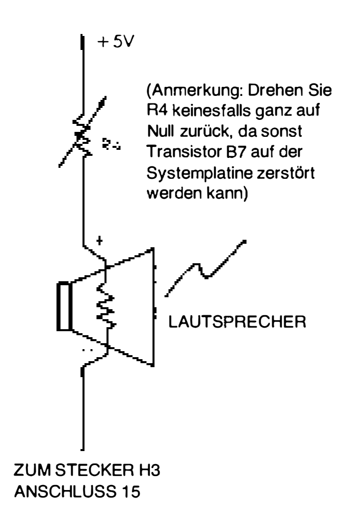


Bild 5.25: Anschluß des Lautsprechers

Die Verdrahtung sehen Sie in Bild 5.25. In Serie zum Lautsprecher sollte zur Strombegrenzung noch ein Widerstand von 50 Ohm oder mehr geschaltet werden. Der Lautsprecher ist an den gepufferten Ausgang des SYM angeschlossen. Wenn man das Potentiometer ohne einen solchen Schutzwiderstand ganz auf Null zurückdreht, können sowohl das Potentiometer als auch der Ausgangstransistor auf der SYM-Platine durchbrennen.

Die Technik zur Tonerzeugung ist meist die gewöhnliche Rechtecksignalmethode, die mit einer Verzögerungsroutine durchgeführt wird.

Anschluß: Steckerleiste A an Stecker H2 anschließen
Steckerleiste AA an Stecker H3 anschließen

Das Programm steuert den Lautsprecher mit einer vorgegebenen Frequenz, die vor der Ausführung des Programms in den Speicher 0004 geladen werden muß.

```
0230  A9 80        TON        LDA #$80
0232  8D 02  AC               STA  $AC02     Setze DDRB des VIA #3
                                             = 80 (Ausgabe an Lautsprecher)
0235  A9 00        WDH        LDA #$00
0237  8D 00  AC               STA  $AC00     Setze Lautsprecherausgang
                                             = „0“ (Lautsprecher ein)
023A  20 48  02               JSR  DLYB      Verzögerungsroutine
023D  A9 80                   LDA #$80
023F  8D 00  AC               STA  $AC00     Setze Lautsprecherausgang
                                             = „1“ (Lautsprecher aus)
0242  20 48  02               JSR  DLYB      Verzögerungsroutine
0245  4C 30  02               JMP  WDH       von vorne
```

Unterprogramm DLYB: dieses Unterprogramm ähnelt der Routine DLYA. Es gibt folgende Unterschiede:

1) die Verzögerung ist wesentlich kürzer
2) die Routine holt den Zähler aus dem Speicher 0004 (der Zähler soll einen negativen Wert haben).

```
0248  A6 04        DLYB       LDX $04
024A  E8           SCHLEIFEINX
024B  E0 00                   CPX #$00
024D  D0 FB                   BNE SCHLEIFE
024F  60                      RÜCKSPRUNG
```

Bild 5.26: Einfache Lautsprecheransteuerung (Programm 5.5)

Der Verzögerungsparameter für dieses Programm muß vor der ersten Ausführung in den Speicher 0004 geladen werden. Er steuert die Frequenz des erzeugten Tons. Bild 5.26 zeigt das Programm. Das Datenrichtungsregister B wird so geladen, daß Bit 7 ein Ausgabekanal ist:

```
TON          LDA  #$80       „10000000“
             STA  #$AC02     DDRB
```

Dann wird der Lautsprecher eingeschaltet:

```
WDH         LDA  #$00
            STA  $AC00
```

Der Lautsprecher bleibt für eine Zeit, die durch den Inhalt des Speichers 0004 festgelegt wird, eingeschaltet, indem das Unterprogramm DLYB aufgerufen wird:

```
            JSR  DLYB
```

Dann muß der Lautsprecher wieder ausgeschaltet werden. Dazu wird Bit 7 des IORB auf „1“ gesetzt:

```
            LDA  #$80
            STA  $AC00
```

Nun muß der Lautsprecher für die gleiche Zeit ausgeschaltet bleiben. Hierzu wird wieder die Routine DLYB aufgerufen:

```
            JSR  DLYB
```

Das Programm springt dann an seinen Anfang zurück:

```
            JMP  WDH
```

Die Verzögerungsroutine DLYB ist im wesentlichen dieselbe wie die Verzögerungsroutine DLYA im Programm 5.1:

```
DLYB        LDX  $04        VERZÖGERUNGS-
                            PARAMETER
SCHLEIFE    INX             ZÄHLER
            CPX  #$00
            BNE  SCHLEIFE
            RTS
```

Wir wollen die durch diese Verzögerungsroutine eingeführte Verzögerungszeit berechnen. Die Dauer jedes Befehls wird rechts neben dem Befehl in Taktzyklen angegeben:

```
                                        Taktzyklen
            LDX  $04                (2)
SCHLEIFE ┌→ INX                     (2)
         │  CPX  #$00               (2)
         └─ BNE  SCHLEIFE           (3)
            RRTS                    (6)
```

Zusätzlich müssen wir die Instruktion JSR (Unterprogrammaufruf), mit der dieses Unterprogramm aufgerufen werden muß, mit 6 Taktzyklen berücksichtigen. Steht im Speicher 0004 beispielsweise der Wert 4, dann wird die Schleife 256 − 4 = 252 mal durchlaufen.

Die Gesamtdauer der Verzögerung ist dann:

$$6 + 2 + (2 + 2 + 3) \times 252 + 6 = 14 + 7 \times 252 = 1778\ \mu sec.$$

Übungsaufgabe 5.13: *Ändern Sie die Verzögerungsroutine so, daß Sie statt eines Inkrementierbefehls einen Dekrementierbefehl verwenden.*

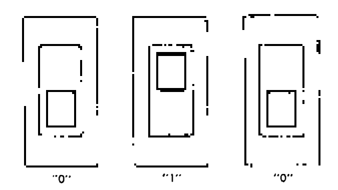


Bild 5.27: Binäre Schalter geben einen Ton vor

Musik

Die Standardmethode zur Erzeugung eines Tones bestimmter Frequenz haben wir bereits vorgestellt. Wir wollen nun einen Schritt weitergehen und eine Melodie spielen. Das Programm wird den binären Wert dreier Schalter A1 bis A3 einlesen und einen Ton erzeugen, der von der Schalterstellung abhängt (siehe Bild 5.27). Die Note c“ (do) wird bei der Schaltstellung „0“ erzeugt, die Note d“ (re) bei einer „1“, usw. Mit drei Schaltern kann man also eine ganze Oktave plus eine Note, d. h. vom c“ bis zum c“’, spielen. Das Programm wird das vorhergehende Programm als Unterprogramm aufrufen. Vor der ersten Benutzung muß der Inhalt des Speichers 0245 von „4C“ in „60“ umgeändert werden. Als erstes werden wir eine Frequenztabelle erstellen, die die jeweilige halbe Periode der gewünschten Rechtecksignale festlegt. Diese Tabelle sehen Sie in Bild 5.28.

```
0050  A2 80      NOTEN   LDX  #$80    Frequenz für c“ (523,3 Hz)
0052  4C 74  02          JMP  LD04
0055  A2 90              LDX  #$90    Frequenz für d“ (587,3 Hz)
0057  4C 74  02          JMP  LD04
005A  A2 9C              LDX  #$9C    Frequenz für e“ (659,3 Hz)
005C  4C 74  02          JMP  LD04
005F  A2 A4              LDX  #$A4    Frequenz für f“ (698,5 Hz)
0061  4C 74  02          JMP  LD04
0064  A2 B0              LDX  #$B0    Frequenz für g“ (784,0 Hz)
0066  4C 74  02          JMP  LD04
0069  A2 B8              LDX  #$B8    Frequenz für a“ (880,0 Hz)
006B  4C 74  02          JMP  LD04
006E  A2 C0              LDX  #$C0    Frequenz für b“ (923,3 Hz)
0070  4C 74  02          JMP  LD04
0073  A2 C4              LDX  #$C4    Frequenz für c“’ (1046,5 Hz)
0075  4C 74  02          JMP  LD04
```

Bild 5.28: Frequenztabelle für Musik

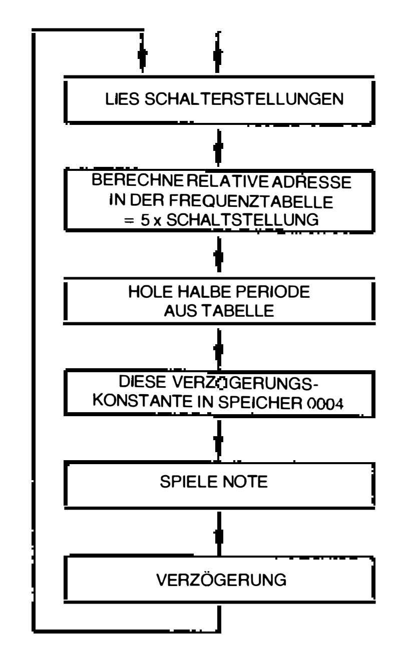


Bild 5.29: Musikprogramm, Flußdiagramm

Verbindung: Steckerleiste A an Stecker H2 anschließen
Steckerleiste AA an Stecker H3 anschließen
Das Programm liest Schalter A1 bis A3 und steuert den Lautsprecher mit 8 verschiedenen Frequenzen an, die durch die Schalter bestimmt werden.
Das Programm verwendet Programm 5.5 als Unterprogramm. Vor Ausführung Speicher 0245 von 4C in 60 ändern.

Das Programm springt zur Tonerzeugung in eine Frequenztabelle. Diese muß vor der Ausführung zuerst geladen werden:

```
0250  A9 00        MUSIK     LDA  #$00      Lade die 8 höherwertigen Bits
                                            für indirekte Sprungadresse
0252  85 05                  STA  $05       Speicher 0005 = 00, da Tabelle
                                            auf Seite Null
0254  8D 03 A0               STA  $A003     Setzt DDRA des VIA #1 = 00
                                            für Eingabemodus
0257  A0 C0        EINGA     LDY  #$C0      (Y) = Verzögerungskonstante
0259  AD 01 A0               LDA  $A001     Lies Schalterstellung
025C  29 07                  AND  #$07      Unterdrücke obere 5 Bits
025E  85 04                  STA  $04       Rette Schalterstellung nach $04
0260  18                     CLC
0261  65 04                  ADC  $04       Berechnung der relativen
0263  65 04                  ADC  $04       Adresse innerhalb der
0265  65 04                  ADC  $04       Frequenztabelle
0267  65 04                  ADC  $04
0269  85 04                  STA  $04
026B  A9 50                  LDA  #$50      Addiere die Anfangsadresse
                                            der Noten-Tabelle dazu
026D  65 04                  ADC  $04
026F  85 04                  STA  $04       Speichere berechnete Adresse
                                            nach 0004 (niederwertig)
0271  6C 04 00               JMP  ($0004)   Springe indirekt in Tabelle
0274  86 04        LD04      STX  $04       Frequenzkonstante nach 04
0276  20 30 02     SCHLEIFEJSR    TON       Unterprogramm Tonerzeugung
0279  88                     DEY
027A  C0 00                  CPY  #$00      Schleife, bis (Y) = 0, dann
                                            neue Schalterabfrage
027C  D0 F8                  BNE  SCHLEIFETon wiederholen
027E  4C 57 02               JMP  EINGA     Schalter erneut abfragen
```

Bild 5.30: Das Musikprogramm (Programm 5.6)

Das Flußdiagramm für diesen Algorithmus zeigt Bild 5.29. Das Programm liest die Stellungen der drei Schalter ein und berechnet dann die relative Adresse innerhalb der Tabelle, wo die benötigte Verzögerungskonstante steht.

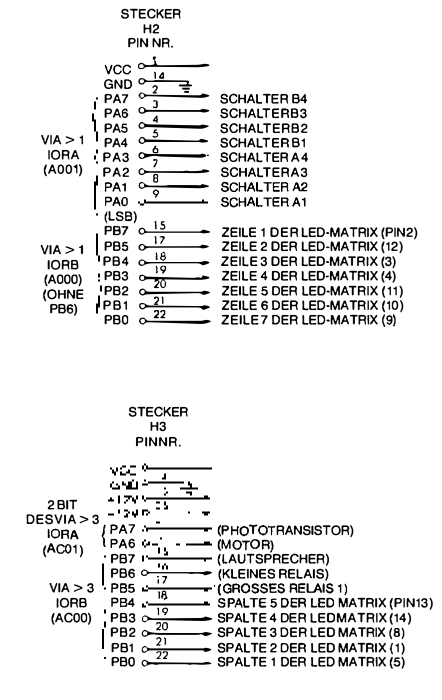


Bild 5.31: Verbindungen für das Musikprogramm

Diese relative Adresse innerhalb der Tabelle entspricht gleich dem fünffachen Wert, der an den Schaltern eingestellt ist. Aus der Tabelle wird die halbe Periode des Rechtecksignals geholt und der Ton für eine gewisse Zeit gespielt. Das Programm wiederholt sich anschließend, so daß die nächste Note (oder nochmals dieselbe) gespielt wird. Das Programm ist in Bild 5.30 gezeigt, das Anschlußschema in Bild 5.31. Die Speicher 04 und 05 werden für einen indirekten Sprung benötigt. Da die Frequenztabelle auf der Seite Null liegt, wird der Inhalt des Speichers 05 gleich zu Anfang auf 00 gesetzt:

```
MUSIK       LDA  #$00
            STA  $05
```

Das Datenrichtungsregister wird dann mit „00“ geladen und legt damit die Eingabefunktion für dieses Tor fest:

```
            STA  $A003
```

Die Dauer des Tons wird durch den Inhalt des Y-Registers festgelegt. Es ist der Zähler für eine äußere Verzögerungsschleife (wir erklären das später noch ausführlicher):

```
EINGA          LDY  #$C0
```

Der Wert der drei Schalter A1 bis A3 wird dann vom IORA mit der Adresse A001 geladen und die oberen 5 Bits maskiert (auf 0 gesetzt):

```
               LDA  $A001
               AND  #$07
```

Die Schalterstellung wird dann in den Speicher 04 gerettet, so daß der Akkumulator für andere Zwecke weiterverwendet werden kann:

```
               STA  $04
```

Um die relative Adresse innerhalb der Frequenztabelle zu berechnen, wird der von den Schaltern gelesene Wert mit 5 multipliziert. Wir machen dies hier durch viermaliges Hinzuaddieren dieses Wertes zu sich selbst:

```
               ADC  $04
               ADC  $04
               ADC  $04
               ADC  $04
               STA  $04
```

Die so errechnete relative Adresse wird dann in den Speicher 04 abgelegt und wir können nun dazu übergehen, aus der Frequenztabelle die halbe Periode zu laden:

```
               LDA  #$50       Anfangsadresse der Tabelle
               ADC  $04        Addiert relative Adresse inner-
                               halb der Tabelle hinzu
               STA  $04
               JMP  ($0004)    Indirekter Sprung
LD04           STX  $04        Frequenzkonstante nach 04
```

Beim Rücksprung aus der Frequenztabelle steht die Verzögerungskonstante im X-Register. Sie wird in den Speicher 04 gelegt und das Unterprogramm TON wird zur Ansteuerung des Lautsprechers aufgerufen:

```
SCHLEIFE       JSR  TON
```

Wie oft der Lautsprecher aktiviert wird, legt der Inhalt des Y-Registers fest:

```
DEY
CPY  #$00
BNE  SCHLEIFE
```

Zum Schluß werden die Schalter nach Erzeugung des Tons für eine bestimmte Zeitspanne erneut abgefragt:

```
JMP  EINGA
```

Wir wollen nun auch an diesem Programm einige Verbesserungen durchführen:

Übungsaufgabe 5.14: *Man könnte die Frequenztabelle dadurch vereinfachen, daß man nur die Verzögerungskonstanten abspeichert, also* $80, $90, etc. Ändern Sie das obige Programm so, daß der von den Schaltern eingelesene Wert als Index zum Laden der Verzögerungskonstanten aus dieser neuen Tabelle verwendet wird. Beachten Sie insbesondere die deutliche Verbesserung in der Gesamtlänge des Programms.

Übungsaufgabe 5.15: *Wenn Sie dieses Programm tatsächlich auf dem Mikrocomputer ausprobieren, werden Sie ein kleines Problem bemerken: das Programm spielt die gewünschte Note korrekt; man hört jedoch gleichzeitig einen überlagerten tieferen Ton. Wenn Sie die letzten 5 Befehle des Programms sorgfältig studieren, sollten Sie herausfinden können, wo dieses Problem liegt. Können Sie ein abgeändertes Programm vorschlagen, bei dem dieser Effekt nicht mehr auftritt? (Ein kleiner Tip: der Lautsprecher wird gelegentlich „zu lange" ausgeschaltet.)*

Übungsaufgabe 5.16: *Sehen Sie sich die Befehle „ADC $04" an, die viermal wiederholt werden. Suchen Sie einen Weg, wie man dasselbe Ergebnis mit möglichst weniger Befehlen erreichen kann.*

Übungsaufgabe 5.17: *Der drittletzte Befehl heißt „CPY #$00". Ist er unbedingt nötig?*

Die Frequenztabelle, die im Musikprogramm verwendet wird, wurde „nach Gehör" und nicht durch Errechnen der korrekten Frequenzkonstanten erstellt. Wir wollen diese jetzt aber überprüfen, um zu sehen, wie gut unsere Tabelle ist.

Seit 1939 ist der Kammerton a' mit 440 Hz festgelegt. Die Frequenzen der anderen Noten verdoppeln sich nach jeder zwölften Halbnote. Für die Frequenzen zweier aufeinanderfolgenden Töne (Halbnoten) gilt daher: $N_2 = \sqrt[12]{2} \times N_1$

Die Frequenzen entnehmen wir Bild 5.28.

Übungsaufgabe 5.18: *Sehen Sie sich die Routine TON an und berechnen Sie die Umschaltzeit in Taktzyklen. Entnehmen Sie dem Bild 5.28 die korrekten Frequenzen und berechnen Sie daraus die theoretischen Verzögerungskonstanten. (Hinweis: Vergessen Sie dabei nicht, daß der Lautsprecher jeweils für eine halbe Periode ein- bzw. ausgeschaltet ist).*

Eine Alarmanlage gegen Einbrecher

Wir wollen eine einsetzbare Alarmanlage für unser Haus entwickeln. Das Eindringen einer Person wird mit einer Lichtschranke (Phototransistor als Detektor) erkannt. Wir wollen annehmen, daß der Lichtemitter stets eingeschaltet ist. Der Detektor wird jede Unterbrechung der Lichtschranke erkennen und den Alarm auslösen. Dieser Alarm wird im Lautsprecher einen Sirenenton erzeugen. Weitere Verbesserungen des Programms werden wir später angeben.

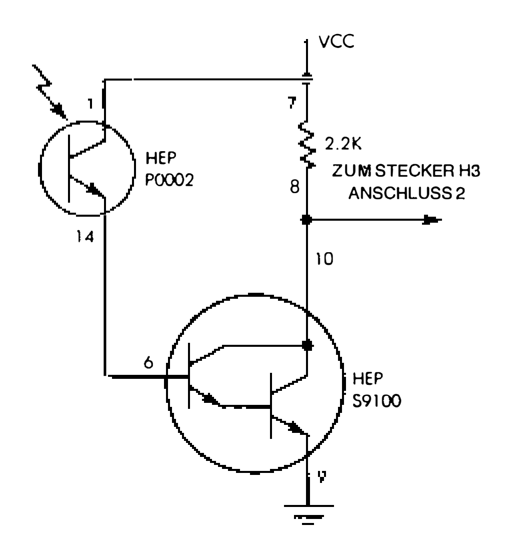


Bild 5.32: Die Schaltung des Phototrausistors (auf Sockel M3)

Der Anschluß des Phototransistors ist in Bild 5.32 gezeigt, das Flußdiagramm für den Algorithmus sehen Sie in Bild 5.33. Wir lesen den Status des Detektors ein. Solange dieser „EIN" ist, hat niemand die Lichtschranke durchbrochen und wir lesen den Status erneut ein. Sobald je-

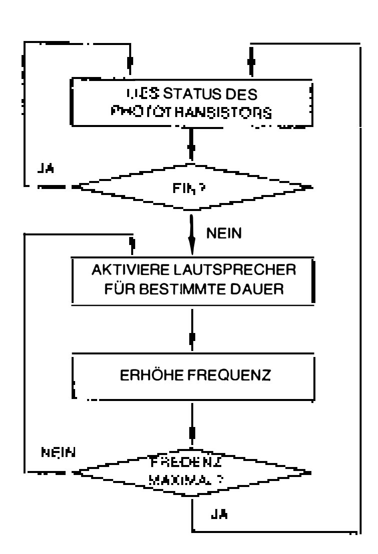


Bild 5.33: Alarmanlage, Flußdiagramm

doch die Lichtschranke durchbrochen wird, zeigt der Detektor den Status „0“ („AUS“) und der Lautsprecher wird für eine bestimmte Zeit aktiviert. Um einen sirenenähnlichen Ton zu erhalten, wird die Frequenz des Alarmtones schrittweise erhöht, bis eine maximale Frequenz erreicht ist (siehe Bild 5.34). An dieser Stelle wird der Status der Lichtschranke erneut abgefragt und die Sirene heult solange, wie der Status „0“ ist. Bild 5.35 zeigt das Programm. Der Ausgang des Phototransistor-Detektors ist in Bit 7 des IORA des VIA #3 angeschlossen (siehe Bild 5.32).

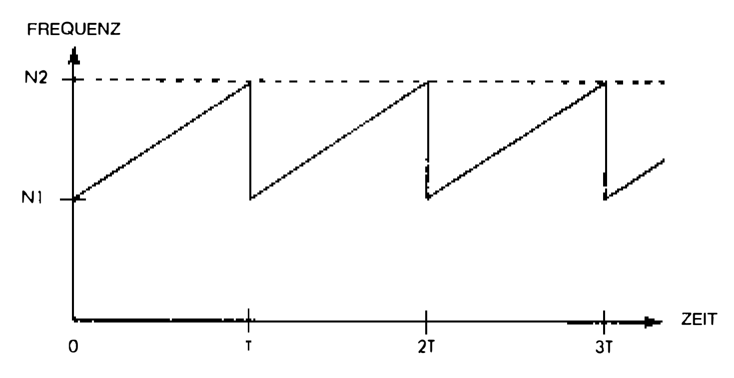


Bild 5.34: Ein Sirenenton

Verbindungen: Steckerleiste A mit Stecker H2 verbinden
Steckerleiste AA mit Stecker H3 verbinden

Das Programm fragt den Ausgang des Phototransistors ab. Wenn dieser „0“ ist, der Phototransistor also beleuchtet ist, geschieht nichts. Ist der Ausgang jedoch auf „1“, was bedeutet, daß der Phototransistor kein Licht mehr sieht, geht sofort der Sirenenton an. Soll die Reaktion entgegengesetzt sein (Alarm bei „0“), so muß BNE durch BEQ ersetzt werden.
Das Programm benutzt das Unterprogramm TON, es muß daher dessen Adresse 0245 in $60 geändert werden.

```
0281  A9 00        ALARM   LDA  #$00
0283  8D 03  AC            STA  $AC03    Setzt DDRA des VIA #3 = 00
                                         für Eingabemodus
0286  AD 01  AC    DETECT  LDA  $AC01    Lädt Status Phototransistor
0289  29 80                AND  #$80     Maskiert Bits 0 bis 6
028B  C9 80                CMP  #$80
028D  F0 F7                BNE  DETECT   Wenn Status = 0 weiter abfragen,
028F  A9 80                LDA  #$80     andernfalls Alarmsirene
                                         initialisieren
0291  85 04                STA  $04      Frequenzkonstante 80 nach 0004
0293  A0 F0        LP7     LDY  #$F0     Verzögerungskonstante nach Y
0295  20 30  02    SIRENE  JSR  TON      Tonerzeugung
0298  C8                   INY
0299  C0 00                CPY  #$00
029B  30 F8                BMI  SIRENE   Schleife, bis (Y) = 0, dann
                                         ändern der Frequenz
029D  A9 01                LDA  #$01     Inkrementiere Frequenz
029F  18                   CLC
02A0  65 04                ADC  $04
02A2  85 04                STA  $04
02A4  C9 A8                CMP  #$A8     Maximale Frequenzkonstante
                                         = A8
02A6  30 EB                BMI  LP7
02A8  4C 86  02            JMP  DETECT   Frage Phototransistor erneut ab
```

Bild 5.35: Alarmanlage (Programm 5.7)

Die ersten Befehle des Programms stellen eine Abfrageschleife dar, mit der der Status des Phototransistors getestet wird:

```
ALARM     LDA  #$00
          STA  $AC03
DETECT    LDA  $AC01
          CMP  #$80
          BNE  DETECT
```

Sobald der Status des Photodetektors „0“ ist (man kann das auf der Experimentalplatine erreichen, indem man den Phototransistor mit dem Fin-

ger oder einem Stück Stoff abdeckt), wird der Alarm ertönen. Die festgelegte Start-Frequenzkonstante wird in den Speicher 04 geladen, die Tondauer in das Y-Register. Die früher entwickelte Routine TON wird dann zur Aktivierung des Lautsprechers aufgerufen:

```
                LDA  #$80
                STA  $04
LP7             LDY  #$F0
SIRENE          JSR  TON
                INY
                CPY  #$00
                BMI  SIRENE
```

Das Unterprogramm TON wird so oft aufgerufen, wie das Y-Register angibt. Die Frequenzkonstante wird dann um 1 vergrößert, in den Speicher 04 zurückgeschrieben und mit der maximalen Frequenz verglichen. Solange diese noch nicht erreicht ist, erzeugt das Programm weiter einen Ton wachsender Frequenz.

```
                LDA  #$01
                CLC
                ADC  $04
                STA  $04
                CMP  #$A8
                BMI  LP7
                JMP  DETECT
```

Wenn die maximale Frequenz erreicht ist, springt das Programm an seinen Anfang zurück. Es sind noch etliche Verbesserungen möglich.

Bei einer realistischen Heimanwendung wird die Alarmanlage irgendwo im Haus installiert sein, und die Lichtschranke, die einen Lichtsender und einen Empfänger umfaßt, wird an irgendeiner anderen Stelle im Haus sein. (In der Praxis verwendet man oft einen Infrarotstrahl, da dieser für das Auge unsichtbar ist.) Man kann sie so anbringen, daß sie einen Raum oder einen Eingang überwacht. Man sollte das Programm dahingehend verbessern, daß man das Haus nach dem Einschalten der Alarmanlage noch verlassen kann, ohne einen Alarm auszulösen. Diese Verbesserung erreichen wir in der ersten Übungsaufgabe:

Übungsaufgabe 5.19: *Ändern Sie das Programm so ab, daß der Benutzer das Haus innerhalb der ersten zwei Minuten nach Einschalten der Anlage noch verlassen kann. Mit anderen Worten: innerhalb der ersten zwei Minuten nach Starten des Programms soll kein Alarm ausgelöst werden, unabhängig vom Status des Photodetektors. Danach soll die Alarmanlage normal funktionieren.*

Dann muß noch ein weiteres Problem gelöst werden: Auch beim Wieder-

betreten des Hauses soll der Alarm nicht sofort losheulen. Vielmehr wollen wir ausreichend Zeit haben, zum Mikrocomputer zu laufen und ihn auszuschalten. Das berücksichtigen wir in der nächsten Aufgabe:

Übungsaufgabe 5.20: *Sobald ein Alarm ausgelöst ist (nach der Einschaltverzögerung von 2 Minuten), soll die Sirene noch 30 sec ausgeschaltet bleiben und erst dann losheulen.*

Wir wollen noch eine weitere Verbesserung einführen: die Lichtschranke kann gelegentlich Störimpulse empfangen, die jedoch die Alarmanlage noch nicht auslösen sollen.

Übungsaufgabe 5.21: *Ändern Sie das Programm derart, daß der Alarm nur ausgelöst wird, wenn die Lichtschranke für länger als 0.05 sec unterbrochen wird.*

Verbessern wir noch weiter: Falls ein Tier die Alarmanlage auslöst, wollen wir ein automatisches Abschaltsystem in Betrieb setzen. Der Alarmton soll nur für zwei Minuten nach Erkennen des Alarms eingeschaltet sein und sich dann selbst ausschalten.

Übungsaufgabe 5.22: *Ändern Sie das Programm so, daß die Sirene nach Auftreten des Alarms für zwei Minuten läuft und sich dann selbst ausschaltet.*

Zusätzlich wollen wir bei einem Alarm weitere Maßnahmen ergreifen, wie etwa das Licht einzuschalten, oder die Polizei anzuwählen. Das alles können wir einführen, indem wir lediglich ein externes Relais einschalten.

Übungsaufgabe 5.23: *Ändern Sie das Programm weiter, so daß jedesmal, wenn ein Alarm ausgelöst wird, ein externes Relais eingeschaltet wird.*

Beachten Sie, daß diese Änderung auch dann sehr vorteilhaft sein kann, wenn Sie die Polizei nicht automatisch anwählen: Sie können eine Lampe an den Relaisausgang anschließen, so daß, selbst wenn ein Einbrecher Ihr Haus betritt und nach Ertönen der Sirene fluchtartig wieder verläßt, Sie dessen Eindringen doch durch die noch brennende Lampe bei Ihrer Rückkehr sofort erkennen würden.

Übungsaufgabe 5.24: *Könnten wir die Instruktion „CPY* #$00" in Zeile 0299 nicht streichen?

Übungsaufgabe 5.25: *Schließen Sie einen „Panikschalter" an, den Sie jederzeit zum Auslösen eines sofortigen Alarmes drücken können. Ändern Sie die Sirene so, daß Ihre Nachbarn einen „Panikalarm" von einem automatischen Alarm unterscheiden können.*

Steuerung eines Gleichstrommotors

Mit diesem Programm soll ein gewöhnlicher Gleichstrommotor gesteuert werden. Ein billiger 12V-Gleichstrommotor für Hobbyanwendungen wird an den Mikrocomputer angeschlossen und die Drehzahl wird mit Schaltern vorgewählt. Wir werden drei Schalter verwenden, so daß 8 verschiedenen Kombinationen möglich sind, die wiederum 8 verschiedene Geschwindigkeiten entsprechen. Die Verdrahtung des Motors sehen wir in Bild 5.36. Den Anschluß der Schalter zeigen Bilder 5.27 und 5.5a.

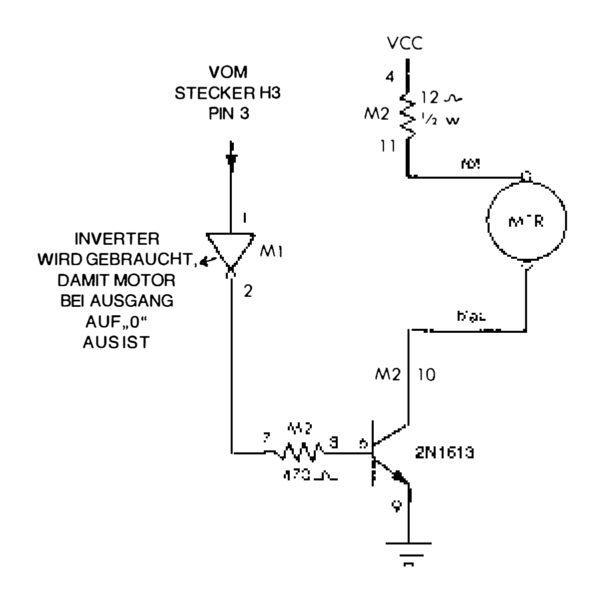


Bild 5.36: Anschluß des Motors

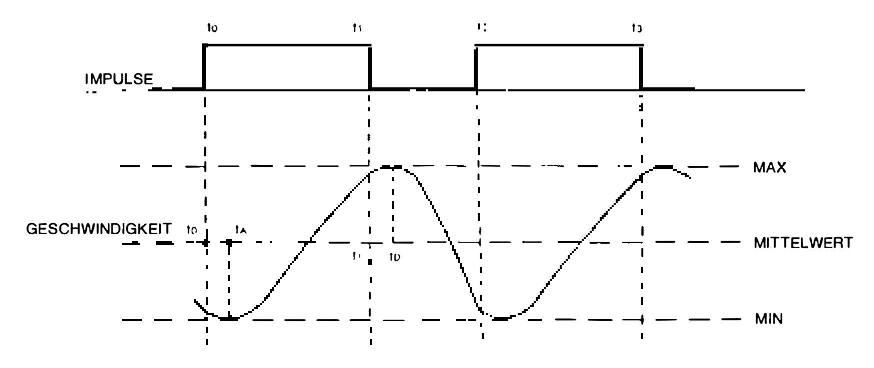


Bild 5.37: Digitale Drehzahlsteuerung

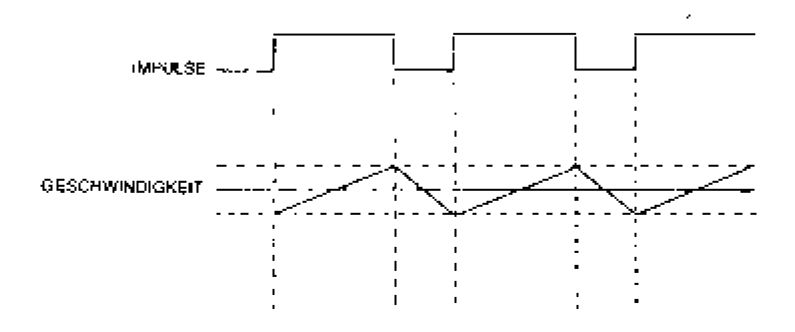


Bild 5.38: Vereinfachtes Geschwindigkeitsdiagramm

Das Prinzip der Drehzahlsteuerung des Motors ist, diesen abwechselnd für eine vorgegebene Zeit ein- und dann wieder auszuschalten. Wegen seiner Drehträgheit wird der Motor noch eine Weile weiterlaufen. Dann wird ein neuer Impuls erzeugt und der Motor wieder eingeschaltet. Dadurch wird er wieder beschleunigt. Das wiederholt sich laufend. Die resultierende Motorgeschwindigkeit zeigt Bild 5.37. Ein vereinfachtes Diagramm derselben Kurve zeigt Bild 5.38. Im wesentlichen ist es eine Sägezahnkurve, wobei der Motor solange beschleunigt, wie die Spannung eingeschaltet ist, und dann bis zum Eintreffen des nächsten Impulses wieder langsamer wird. Im Bild 5.37 wird die mittlere Geschwindigkeit durch die horizontale Linie zwischen der maximalen und der minimalen Geschwindigkeit angedeutet. Man erkennt im Diagramm, daß die Geschwindigkeit dauernd zwischen dem minimalen Wert hin- und herpendelt. Soll die mittlere Geschwindigkeit mit hoher Genauigkeit eingehalten werden, dann müssen minimale und maximale Geschwindigkeit dicht beieinander liegen. Dies erreicht man durch die Wahl kurzer Impulse. Jedoch treten in dem System wie immer, wenn Trägheitseffekte und Oszillationen vorhanden sind, Instabilitäten auf. In unserem Diagramm bedeutet das, daß die Drehzahl nicht kleiner wird, wenn der nächste Einschaltimpuls vor Ablauf der Zeitspanne t_D eintrifft, und daß dann die Geschwindigkeit nicht kleiner, sondern laufend größer wird. Das liegt an der Trägheit des beschleunigenden Magnetfeldes im Motor, das noch nicht so weit abklingen konnte, daß der Motor schon wieder langsamer würde. Andere, noch komplexere Effekte können auftreten. Diese Fragen wollen wir hier jedoch nicht erörtern. Wir werden ganz einfach ein Programm erstellen, bei dem wir die einzelnen Verzögerungszeiten einstellen können und diese dann später durch Ausprobieren so bestimmen, daß das Programm mit dem von uns gewählten Motortyp funktioniert. Der Leser sollte lediglich im Gedächtnis behalten, daß diese Verzögerungskonstanten auf verschiedene Wege gewählt werden können, so daß die Konstanz der Drehzahl verbessert und/oder Probleme mit Oszillationen beseitigt werden können.

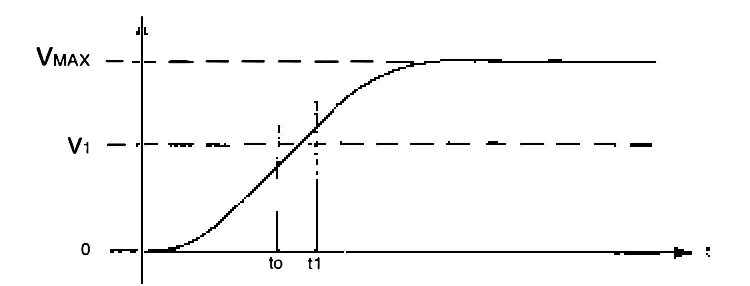


Bild 5.39: Geschwindigkeitskurve des Gleichstrommotors

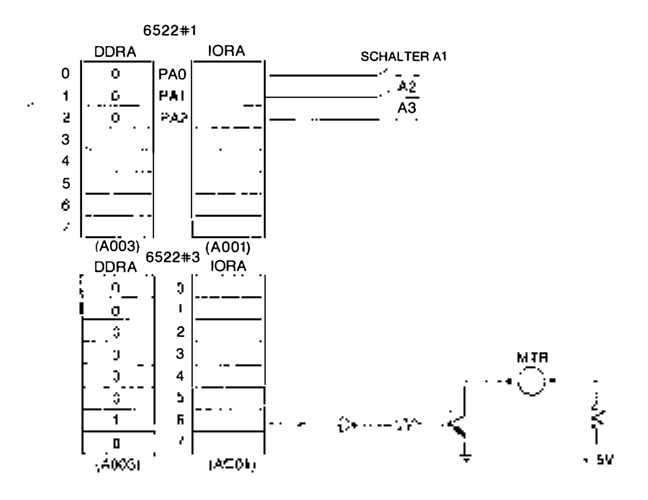


Bild 5.40: Das Anschlußbild

Die Hardware

Es werden zwei Tore verwendet: das des 6522#1 und das des 6522#3. Die Anschlüsse werden in Bild 5.40 gezeigt. Das obere IORA-Register wird als Eingabetor für die drei Schalter verwendet. Die Schalterstellung wird die Geschwindigkeit des Motors festlegen. Die entsprechenden Bits des DDRA sehen Sie links im Bild. Das untere IORA (VIA#3) wird als Ausgabetor zur Steuerung des Motors eingesetzt. Der Motor ist an Bit 6 des IORA angeschlossen. Einen detaillierten Schaltplan des Interfaces entnehmen Sie Bild 5.36. Der Inverter wird zum Negieren des Signals gebraucht, und der Transistor, um eine ausreichende Stromstärke zur Verfügung zu stellen.

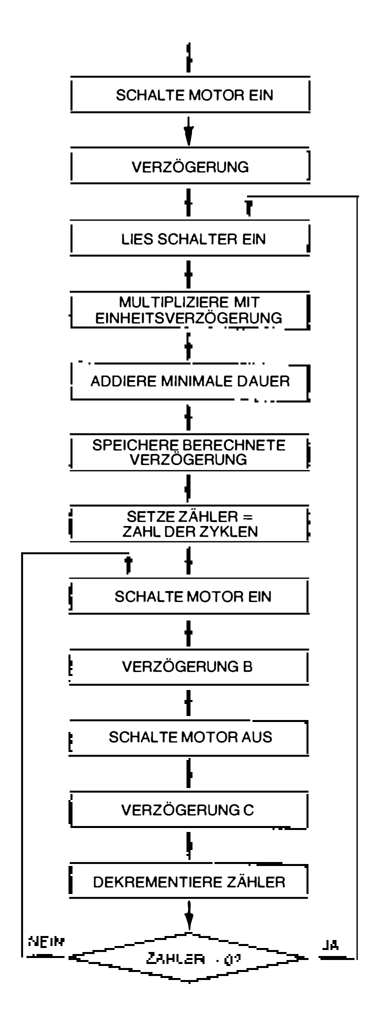


Bild 5.41: Gleichstrommotor, Flußdiagramm

Das Programm

Das Flußdiagramm für das Programm ist in Bild 5.41 gezeigt. Der Motor wird für eine Zeit T_{ein} eingeschaltet und dann für eine Zeit T_{aus} ausgeschaltet. In diesem Algorithmus ist die Zeit T_{aus} festgelegt und die Zeit T_{ein} für jede Schalterstellung von „000" bis „111" ansteigend. Der Schalterstellung „000" entspricht eine minimale Zeit für T_{ein}. Die Verzögerungszeit, die einer bestimmten Schalterstellung entspricht, kann mit folgender Formel berechnet werden:

$$T_{ein} = \text{MIN} + \text{Einheitsverzögerung x Schalter.}$$

Die Zahlenwerte für die Verzögerungskonstanten sind:

$$\text{VERZÖGERUNG}_{aus} = \$C0 = 192 \text{ dezimal}$$
$$\text{VERZÖGERUNG}_{ein} = \$80 + \text{SCHALTER X } \$0B$$
$$= 128 + \text{SCHALTER X } 11 \text{ dezimal}$$

SCHALTER	000	001	010	011	100	101	110	111
VERZÖGERUNG ein	128	139	150	161	172	183	197	205

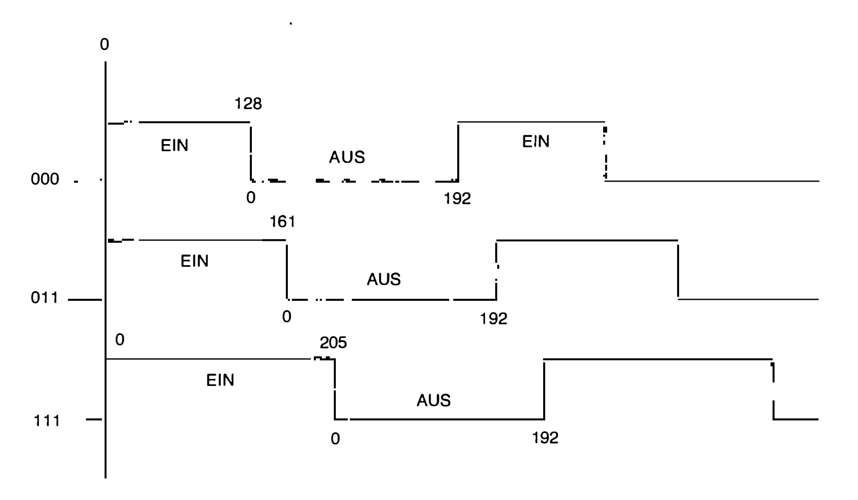


Bild 5.42: Die Fonnen des Ausgangssignals

In Bild 5.42 sehen Sie die Formen des Ausgangssignals, die bei verschiedenen Schalterstellungen erzeugt werden. Wenden wir uns nun dem Flußdiagramm in Bild 5.41 zu. Zuerst wird der Motor für eine gewisse Startzeit eingeschaltet, damit er eine bestimmte Anfangsdrehzahl erreicht (andernfalls kann es passieren, daß eine Kette kurzer Impulse nicht ausreicht, den Motor in Bewegung zu setzen). Dann wird der mit den Schaltern eingestellte Wert eingelesen und die zugehörige Verzögerung muß berechnet werden. Der an den Schaltern eingestellte Wert wird mit einer Verzögerungseinheit multipliziert und um eine Mindestimpulsdauer vergrößert. Die so berechnete Verzögerungskonstante wird abgespeichert. Dann wird der Motor für diese berechnete Verzögerungszeit eingeschaltet. Wir nennen dies Verzögerung B. Dann wird der Motor für die Verzögerungszeit C wieder ausgeschaltet. Dieser Prozeß wird mehrmals wiederholt, damit sich die Drehzahl stabilisieren kann. Dann können die Schalter erneut abgefragt werden und falls sich die Einstellung geändert

hat, wird die neue Geschwindigkeit nachreguliert. Beachten Sie ferner, daß durch die eingebaute Verzögerung, die durch das mehrmalige Wiederholen des Prozesses zustande kommt, auch das Problem des Kontaktprellens entfällt. Wäre diese Verzögerungszeit, während der sich die Geschwindigkeit einstellen kann, nicht vorgesehen, dann müßte man die Kontakte hardwaremäßig oder softwaremäßig entprellen (für Details siehe auch ,Mikroprocessor Interface Techniken').

Verbindungen: Steckerleiste A an Stecker H2
Steckerleiste AA an Stecker H3
Das Programm liest die Schalter A1 bis A3 ein und versorgt den Motor entsprechend der dadurch vorgegebenen Drehzahl mit Strom.
Das Programm ruft die beiden Unterprogramme DLYA und DLYB auf.

```
02B0 A9 40          MOTOR LDA #$40
02B2 8D 03 AC             STA $AC03    Setze DDRA des VIA #3 = 40
                                       für Motorsteuerausgang
02B5 A9 00                LDA #$00     Schalte Motor für Dauer von
                                       DLYA ein für Anfangs-
                                       geschwindigkeit
02B7 8D 01 AC             STA $AC01
02BA A9 FF                LDA #$FF
02BC 85 00                STA $00
02BE 20 20 01             JSR DLYA
02C1 A9 00                LDA #$00     Setze DDRA des VIA #3 = 00
                                       für Eingabemodus
02C3 8D 03 A0             STA $A003
02C6 AD 01 A0 MTRSP       LDA $A001    Lies Schalterstellung
02C9 29 07                AND #$07     Unterdrücke obere 5 Bit
02CB A8                   TAY          (Y)=Schalterstellung
02CC A9 0B                LDA #$0B     Setze Verzögerungsdifferenz
                                       zwischen Stellungen = 0B
02CE 85 06                STA $06
02D0 C0 00       LP8      CPY #$00
02D2 F0 07                BEQ ONDLY
02D4 18                   CLC
02D5 65 06                ADC $06
02D7 88                   DEY          Schleife, bis ($0006)=
                                       Schalterstellung x $0B
02D8 4C D0 02             JMP LP8
02DB 85 06       ONDLY    STA $06
02DD A9 80                LDA #$80     Berechne Verzögerung, ein =
02DF 18                   CLC          S80 + (Schalterstellung x $0B)
02E0 65 06                ADC $06
02E2 85 06                STA $06      Verzögerung, ein nach 0006
02E4 A0 C0                LDY #$C0
```

Bild 5.43: Motorsteuerung (Programm 5.8)

```
02E6  A5  06          MTREIN  LDA  $06      Übertrage (0006) nach 0004
                                            vor Aufruf von DLYB
02E8  85  04                  STA  $04
02EA  A9  00                  LDA  #$00     Schalte Motor ein
02EC  8D  01  AC              STA  $AC01
02EF  20  48  02              JSR  DLYB     Verzögerungsroutine DLYB
02F2  A9  C0                  LDA  #$C0     Setze Verzögerungskonstante
                                            für aus = $C0, unabhängig
                                            von Schalterstellung
02F4  85  04                  STA  $04      Speichere sie nach 0004
02F6  A9  40          MTRAUS  LDA  #$40     Schalte Motor aus
02F8  8D  01  AC              STA  $AC01
02FB  20  48  02              JSR  DLYB     Verzögerungsroutine DLYB
02FE  88                      DEY
02FF  C0  00                  CPY  #$00     Wiederhole ein/aus-Schalten
                                            bis (Y) = 00
0301  30  E3                  BMI  MTREIN
0303  4C  C6  02              JMP  MTRSP    Lies Schalterstellung neu
                                            ein und steuere Motor weiter
```

Bild 5.43: (Fortsetzung)

Das Programm sehen Sie in Bild 5.43. Die ersten vier Instruktionen schalten den Motor ein, indem sie das Datenrichtungsregister laden und eine „0“ in das Datenregister legen:

```
MOTOR       LDA  #$40
            STA  $AC03
            LDA  #$00
            STA  $AC01
```

Dann wird die Verzögerungskonstante „FF“ in den Speicher „00“ gelegt. Dabei wird angenommen, daß vereinbart wurde, daß die Routine DLYA ihre Verzögerungskonstante aus diesem Speicher holt (siehe Programm 5.1). Dann wird das Unterprogramm DLYA aufgerufen. Hierdurch wird eine Anfangsverzögerung erreicht, während der Motor eine Anfangsgeschwindigkeit erreichen kann.

```
            LDA  #$FF
            STA  $00
            JSR  DLYA
```

Nun wird die Schalterstellung eingelesen:

```
            LDA  #$00
            STA  $A003
MTRSP       LDA  $A001
```

Und von dem eingelesenen Wert werden die drei untersten Bit abgetrennt:

```
                AND #$07       MASKE
                TAY
```

Für jede Schalterstellung außer „000" wird zu der Mindestdauer „0B" hexadezimal noch eine zusätzliche Verzögerungseinheit hinzuaddiert. Deshalb wird der Wert der Schalterstellung in das Y-Register gerettet und die Mindestverzögerung in den Speicher 06 geladen.

```
                LDA #$0B
                STA $06
```

LP8 ist eine zusätzliche Schleife, in der die Verzögerungseinheit so oft aufaddiert wird, wie die Schalter angeben:

```
LP8             CPY #$00
                BEQ ONDLY
                CLC
                ADC $06
                DEY
                JMP LP8
```

Wenn das Programm bei ONDLY angelangt ist, enthält der Speicher 06 die zusätzliche Verzögerungskonstante, die der Schalterstellung entspricht. Diese wird dann zur minimalen Verzögerungskonstanten „80" hexadezimal hinzuaddiert:

```
ONDLY           STA $06
                LDA #$80
                CLC
                ADC $06
                STA $06
```

Dann wird das Y-Register mit dem Wert „C0" hexadezimal geladen, der angibt, wie oft der Motor ein- und ausgeschaltet werden soll:

```
                LDY #$C0
```

Bei Erreichen des Programmteils MTREIN steht im Speicher „06" die Verzögerungskonstante, die die Einschaltdauer des Motors bestimmt. Sie wird in den Speicher „04" übertragen, damit das Unterprogramm DLYB verwendet werden kann. Dann wird der Motor eingeschaltet und die Verzögerung abgerufen:

```
MTREIN          LDA $06
                STA $04
                LDA #$00       MOTOR EIN
                STA $AC01
                JSR DLYB
```

Nun muß die Ausschaltperiode folgen. Dazu wird der Wert „C0" hexadezimal in den Speicher 04 geladen. Der Motor wird explizit ausgeschaltet und die Verzögerungsroutine DLYB erneut aufgerufen:

```
              LDA  #$C0
              STA  $04
MTRAUS        LDA  #$40        MOTOR AUS
              STA  $AC01
              JSR  DLYB
```

Nach der Ausschaltperiode wird der Schleifenzähler Y dekrementiert. Wir verwenden das Indexregister Y hier zum Abzählen, wie oft der Ein/Ausschaltzyklus schon durchgeführt wurde. Es wurde mit dem Wert „C0" hexadezimal initialisiert und wird nach jedem Ausschalten des Motors dekrementiert. Nach Erreichen des Wertes „0" springt das Programm wieder an den Anfang und liest die nächste Schalterstellung ein. Falls Y noch nicht bis „0" dekrementiert ist, springt das Programm zurück zu MTREIN und durchläuft den Ein/Ausschaltzyklus erneut:

```
              DEY
              CPY  #$00
              BMI  MTREIN
              JMP  MTRSP
```

Überlegen wir uns nun einige Programmverbesserungen.

Übungsaufgabe 5.26: *Verbessern wir zunächst den Programmierstil ein wenig: Sehen Sie sich das Programm von Zeile 02D0 bis 02D8 an. Fällt Ihnen eine Verbesserung in der Reihenfolge der Befehle ein? (Hinweis: Man kann einen Befehl einsparen)*

Übungsaufgabe 5.27: *Stellen Sie dieselben Überlegungen für die Zeilen 02FF bis 0303 an.*

Übungsaufgabe 5.28: *Falls Sie das Programm tatsächlich mit einem Motor ausprobieren können, ist folgende Überlegung von Wert: erhöhen Sie schrittweise die Verzögerungskonstante für die Ausschaltperiode, indem Sie die entsprechende Konstante im Programm ändern. Was passiert?*

Übungsaufgabe 5.29: *Machen Sie dasselbe bei Verkleinerung der Konstanten. Welches Problem ergibt sich hier?*

Übungsaufgabe 5.30: *Ein anderer Algorithmus, den man anwenden könnte, wäre, daß man eine veränderliche Zahl von Einschaltperioden konstanter Dauer einführt, d. h. man verändert die Dauer der Ausschaltperiode anstatt die der Einschaltperiode. Können Sie das Programm entsprechend ändern?*

Wichtiger Hinweis: Da jeder Motor verschiedene Charakteristiken aufweist, werden die Zeitkonstanten in diesem Programm am besten experimentell bestimmt. Es wird Ihnen dringend empfohlen, die verschiedenen Konstanten, die im Programm vorkommen, wie zum Beispiel die Verzögerungszeitkonstanten für die Ein- oder Ausschaltperioden oder die Schleifenzähler, abzuändern, bis Sie Erfahrungswerte gefunden haben, mit denen Sie die besten Ergebnisse erzielen. Falls Sie den Motor in einer praktischen Anwendung einsetzen wollen, werden Sie zusätzliche Parameter einführen, die die Beschleunigungs- und Reibungseffekte berücksichtigen. Ferner sind billige Hobbymotoren oft schlecht gelagert und schlecht geschmiert. Nach wenigen Wochen oder Monaten kann sich daher der Einfluß der Reibung erheblich vergrößert haben. Sie werden dann eine deutlich längere Anlaufzeit und eventuell auch andere Ein/Ausschalterperiodendauern benötigen. Solange Sie den mechanischen Zustand Ihres Motors im Auge behalten, sollten Sie in der Lage sein, die entsprechenden Parameter den veränderten Verhältnissen anzupassen.

Übungsaufgabe 5.31: *Können Sie voraussagen, was passiert, wenn Sie sehr kurze Einschaltperioden haben?*

Das obige Programm ist ein offener Regelkreis, in dem wir die Geschwindigkeit des Motors zwar steuern, aber nicht messen. Überlegen wir uns mögliche Verbesserungen bei dieser Technik.

Übungsaufgabe 5.32: *Stellen Sie die eingestellte Geschwindigkeit des Motors auf einer Anzeige dar. Die Geschwindigkeitsanzeige könnte identisch mit der Schalterstellung sein, d. h. Sie könnten einfach eine Zahl zwischen 0 und 7 anzeigen.*

In der nächsten Übungsaufgabe wollen wir einen richtigen geschlossenen Regelkreislauf programmieren. Diese Aufgabe ist besonders dann wichtig, wenn Sie die Methoden verstehen wollen, die man beispielsweise bei der Steuerung von Diskettenlaufwerken verwendet. Ein einfacher und wirkungsvoller Weg zur Messung der Motordrehzahl ist, eine Plastikscheibe mit der Welle zu koppeln. In diese Scheibe ist ein Loch gestanzt. Auf der einen Seite der Scheibe wird eine Lichtquelle angeordnet, auf der anderen Seite ein lichtempfindlicher Detektor. Beides wird so justiert, daß Licht auf den Detektor fällt, wenn das Loch vor der lichtemittierenden Diode vorbeidreht. (Genau das macht man in einer Floppy Disk zur Erkennung des Justierloches.) Jedesmal, wenn der Detektor beleuchtet wird, wird ein Impuls erzeugt. Indem man die Anzahl der Impulse pro Sekunde zählt, erhält man die genaue Drehzahl des Motors in Umdrehungen pro Sekunde. Mit Hilfe dieser Information läßt sich die Dauer oder die Frequenz der Ein- oder Ausschaltperioden so einstellen, daß die Motordrehzahl mit hoher Genauigkeit geregelt wird. Wir führen den Vergleich zwischen dieser Technik und einer Floppy Disk hier nicht mehr weiter, da bei einem Diskettenlaufwerk die Drehzahl mit äußerst hoher

Präzision eingehalten werden muß und sogar während Teilumdrehungen der Diskette und nicht nur in ihrem Mittelwert nachgeregelt werden muß. Daher zieht man bei einem Diskettenlaufwerk Zusatzinformationen hinzu: alle Informationen sind in Spuren aufgezeichnet und werden dazu benutzt, die Drehgeschwindigkeit während Bruchteilen einer einzelnen Umdrehung nachzuregeln. Im Falle unseres Motors ist es wichtig, die aktuelle Drehzahl zu messen, da jegliche Reibung oder Belastung des Motors dessen Geschwindigkeit verändern wird. Alle hierfür nötigen Soft- und Hardwaretechniken wurden bereits eingeführt.

Übungsaufgabe 5.33: *Schreiben Sie ein Programm, das dies durchführt.*

Analog-Digital-Wandlung (ein Thermometer)

Wir werden zur Messung der Temperatur einen Thermistor, also einen temperaturabhängigen Widerstand, verwenden. Jedes andere temperaturempfindliche Bauteil kann ebenso verwendet werden. Der Widerstand des Thermistors ändert sich über der Temperatur. Wir werden ihn zur Erfassung von Temperaturänderungen einsetzen und je nach gemessener Temperatur geeignete Maßnahmen ergreifen. Das Hauptproblem bei der Vorgabe eines analogen Wertes (also eines Wertes, der sich kontinuierlich ändert, in unserem Fall der Widerstand des Thermistors) ist die Darstellung des Meßergebnisses als binäre Zahl. Man nennt dieses Problem „Analog-Digital-Wandlung“. Es gibt heutzutage Komponenten, die solche Umwandlungen im wesentlichen mit einem einzigen Baustein durchführen. Wir werden hier einen billigeren (und lehrreicheren) Weg beschreiten. Wir werden einen Digital-Analog-Wandler und einige Operationsverstärker verwenden. Die eigentliche Analog-Digital-Wandlung wird das Programm vornehmen (für Details der analog zu digital Wandlung schlagen Sie bitte im Kapitel 5 des Buches „Mikroprozessor Interface Techniken“ nach).

Wir werden die *Technik der sukzessiven Approximation*, also einer schrittweisen Annäherung, anwenden. Es wird ein binärer Ausgangswert erzeugt und dieser in analoge Form umgewandelt. Dieser erste analoge Näherungswert wird durch eine Komparatorschaltung mit dem vom Thermistor erzeugten Wert verglichen. Das Ergebnis dieses Vergleichsvorganges, also eine „0“ oder eine „1“, je nachdem der Näherungswert größer oder kleiner war, wird beim Erzeugen des nächsten Näherungswertes verwendet.

Die für dieses Experiment nötige Hardware und Verdrahtung sehen Sie in Bild 5.44. Der 8-Bit Ausgang des IORA ist an einen 8-Bit DAC (digital zu analog Wandler) angeschlossen. Dieser DAC wandelt die binäre 8-Bit Zahl in ein Analogsignal um, dessen Wert dann mit dem des Thermistors verglichen wird. Der Komparatorausgang ist wiederum an Bit 0 des IORB angeschlossen, wo er eingelesen werden kann.

Der Algorithmus wird schrittweise jedes Bit des IORA vom höchsten Bit (Bit 7) bis herunter zum niedrigsten Bit (Bit 0) einschalten.

Der erste Wert, der ausprobiert wird, ist „10000000". Wenn dieser Wert als zu klein befunden wird, dann wird Bit 7 ungeändert belassen und Bit 6 wird eingeschaltet. In diesem Fall wird die zweite Näherung also „11000000" sein. Wenn dieser Näherungswert zu groß ist, was durch Abfragen des Komparatorausganges festgestellt wird, wird Bit 6 wieder ausgeschaltet. Der dritte Näherungswert wird in diesem Fall „10100000" sein. Bit 5 wurde automatisch eingeschaltet. Und so weiter.

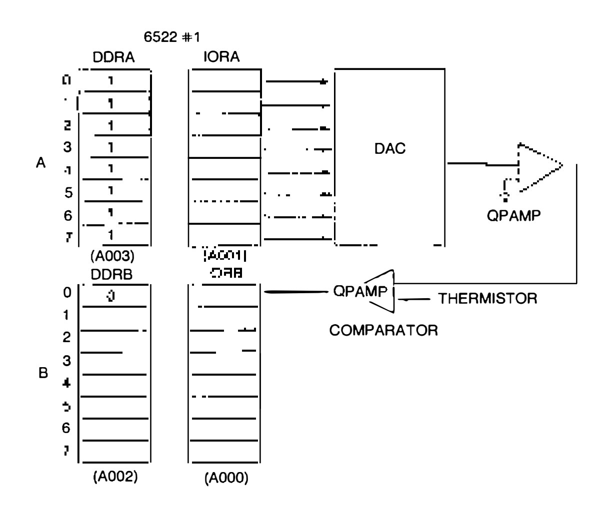


Bild 5.44: Anschluß zur Analog-Digital-Wandlung

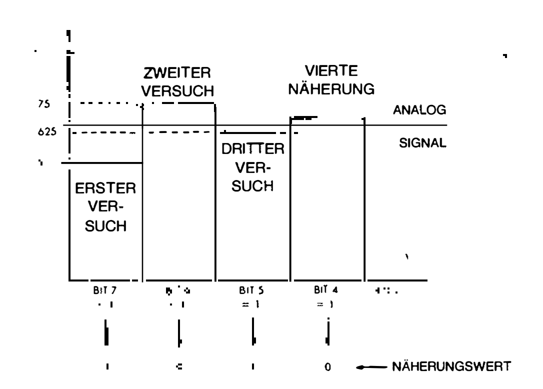


Bild 5.45: Sukzessive Approximation

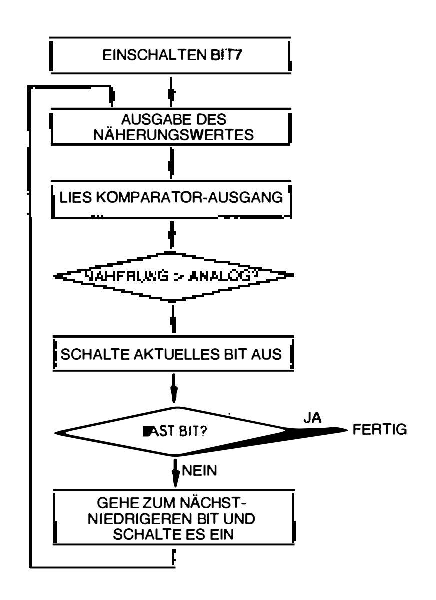


Bild 5.46: Sukzessive Approximation, Flußdiagramm

Bild 5.47: Das A/D-Wandler Interface

Den formalen Algorithmus sehen Sie in Bild 5.45, das Flußdiagramm in Bild 5.46. Der Prozeß wird solange durchgeführt, bis alle 8 Bits durchlaufen sind. Der resultierende binäre Wert ist die bestmögliche Näherung des Analogwertes mit der bei einer 8-Bit Darstellung möglichen Genauigkeit. Natürlich wird bei diesem Prozeß vorausgesetzt, daß der Algorithmus schnell genug ausgeführt wird, so daß sich der Analogwert nicht schneller ändert, als er gemessen wird. Andernfalls sollte eine *sample-and-hold* Schaltung verwendet werden. Die Darstellung im Bild 5.45 zeigt die schrittweise Annäherung an das analoge Signal. Jedes Mal, wenn ein neues Bit hinzugezogen wird, wird das Intervall durch 2 geteilt.

Die Hardware

Die Hardware ist in Bild 5.47 und 5.48 zu sehen. Als D/A-Wandler setzen wir einen MC1408 ein, der eine 12V Spannungsversorgung benötigt. Dessen Ausgang treibt den Operationsvertärker M5, dessen Ausgangssignal weiter zum Komparator geht. Der Thermistor ist unten im Bild zu sehen. Sein Signal speist den anderen Eingang des Komparators. Der Ausgang des Komparators ist an Pin 22 des Steckers H4 angeschlossen. Er steuert Bit 0 des IORB des 6522#1.

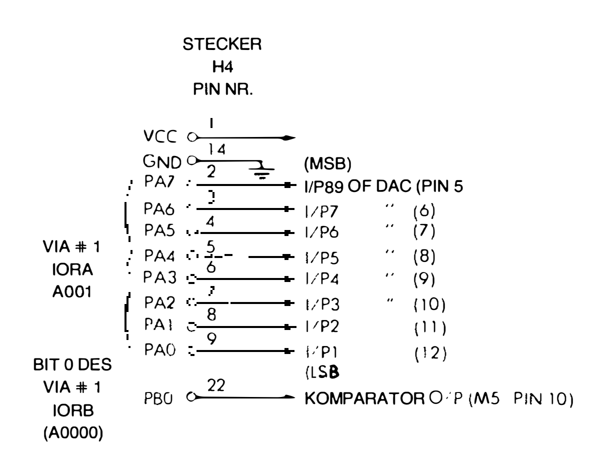


Bild 5.48: Anschluß an Stecker H4

Das Programm

In diesem Programm wird die vom Thermistor gemessene Temperatur durch die Frequenz eines Tones im Lautsprecher angezeigt. Die Tonhöhe wird umso höher, je höher die Temperatur ist.

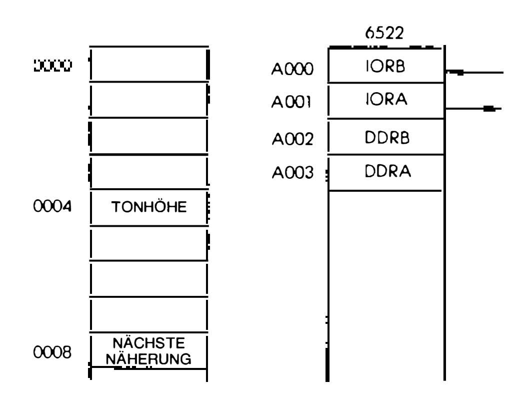


Bild 5.49: Speicherbelegung des A/D-Wandlers

Die Speicherbelegung für das A/D-Wandlungs-Programm ist in Bild 5.49 gezeigt. Der Speicher 4 wird dazu benutzt, die Konstante, die das Unterprogramm DLYB verwendet, zu speichern. DLYB erzeugt eine durch den Wert dieser Konstanten festgelegten Verzögerung. In Speicher 8 wird die vom Programm berechnete jeweilige neueste Näherung abgelegt. Der 6522#1 liegt ab Speicher A000.

Bild 5.50 zeigt das Flußdiagramm. Als erstes wird der 6522 so initialisiert, daß IORA als Ausgabetor für den D/A-Wandler, und das Bit 9 des IORB als Komparatoreingang vereinbart wird. Das Zeigerregister wird auf seinen Startwert von „10000000" gesetzt. Dies ist gleichzeitig auch der erste Näherungswert. Dieses Zeigerregister wird auf jeweils das Bit zeigen, das in der Näherungsschleife gerade eingeschaltet ist. Jedesmal, wenn die Schleife durchlaufen ist, wird das Bit nach rechts geschoben.

Als erste Näherung wird der Inhalt des Zeigerregisters gewählt. Dieser Wert wird dann in ein Analogsignal umgesetzt. Um dem D/A-Wandler ausreichend Zeit für die Umwandlung zu geben, wird eine Verzögerungsschleife eingeführt, und erst nach dieser der Komparatorausgang abgefragt. Falls dieser eine „1" zeigt, dann ist diese neue Näherung zu klein, und der Wert braucht nicht geändert zu werden. Ist der Komparatorausgang „0", so ist der Näherungswert zu groß, und das aktuelle Bit muß wieder zurückgesetzt werden. Als nächstes wird das Zeigerregister um eine Position nach rechts verschoben, damit es auf das nächste bei dieser Technik benötigte Bit zeigt. Falls das letzte Bit erreicht ist, ist die endgültige Näherung errechnet. Falls nicht, dann wird eine neue Näherung erreicht, indem der Wert des Zeigerregisters zur alten Näherung addiert wird. Die Schleife wird dann erneut durchlaufen.

Sobald der endgültige Näherungswert gefunden ist, muß ein Ton erzeugt werden, dessen Höhe von dem gemessenen Wert abhängt. Es wird eine

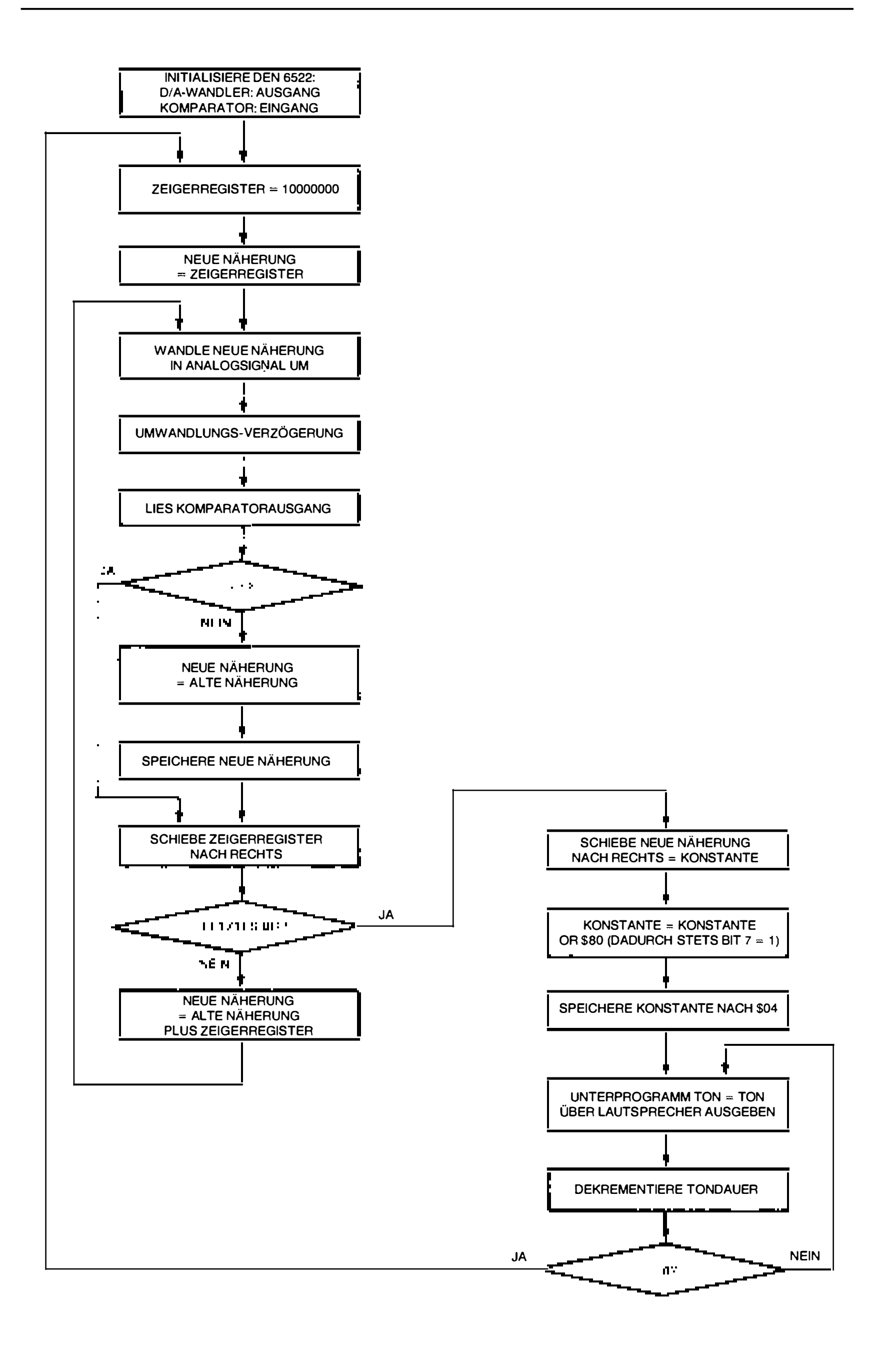


Bild 5.50: Flußdiagramm des Programms A/D-Wandlung

minimale Tonfrequenz vorgegeben. Die Tonhöhenkonstante wird dadurch erzeugt, daß der halbe Näherungswert zu der Tonhöhenkonstanten dieser minimalen Frequenz (= $80) addiert wird. Dann wird die Routine TON aufgerufen, die den Lautsprecher aktiviert. Nachdem der Lautsprecher den Ton für eine gewisse Mindestzeit erzeugt hat, liest das Programm den Wert des Thermistors erneut.

Bei dem beschriebenen Aufbau erhält man eine hörbare Änderung am schnellsten, wenn man einen Lötkolben oder eine Zigarette mit der Spitze nahe an den Thermistor hält. Der Ton aus dem Lautsprecher sollte dabei seine Frequenz schnell erhöhen. Zieht man den Lötkolben bzw. die Zigarette wieder weg, läuft derselbe Vorgang rückwärts ab. Der Thermistor könnte natürlich auch außerhalb der Platine angebracht sein. Bei ausreichender elektrischer Isolation könnte er beispielsweise an einer Wand, in einer Tasse, oder wo sonst Temperaturen gemessen werden sollen, angebracht sein. Man könnte auch ein Thermoelement verwenden, oder dieses in eine Flüssigkeit tauchen, so daß man deren Temperatur messen könnte. Die Umgebungstemperatur könnte gesteuert werden, indem beispielsweise eine Heizspirale mit einem der Relais geschaltet würde. Es würde nur noch das Problem bestehen bleiben, den Thermistor so zu eichen, daß exakte Temperaturmessungen möglich sind.

Verbindungen: Stecker A nach Stecker H4
Stecker AA nach Stecker H3

Dieses Programm verwendet eine schrittweise Näherung mit einem D/A-Wandler, so daß der Analogwert eines Themistors kontinuierlich erfaßt werden kann. Dann wird der angenäherte digitale Wert als Parameter zur Steuerung der Frequenz eines Tones benutzt. Aus der Frequenzänderung kann man auf eine Temperaturerhöhung oder -erniedrigung schließen.

Die Tonfrequenz ist der Temperatur (oder dem Widerstand des Thermistors) proportional.

Das Programm ruft die Unterprogramme TON und DLYB auf.

```
0360  A9 FF      ADC    LDA  #$FF
0362  8D 03 A0          STA  $A003    Setze das DDRA des VIA #1
                                      = $FF als Ausgabetor für den
                                      D/A-Wandler
0365  A9 00             LDA  #$00
0367  8D 02 A0          STA  $A002    Setze das DDRB des VIA #1
                                      = $00 als Eingabekanal zum Lesen
                                      des Komparatorausgangs
036A  A9 80      START  LDA  #$80     Erste Näherung = 10000000
036C  A8                TAY           (Y) = aktuelles Bit
036D  85 08             STA  $08      Speicher $0008 enthält
                                      aktuelle Näherung
```

Bild 5.51: A/D-Wandlungs-Programm (Programm 5.9)

```
036F  A5 08        NXTBIT  LDA  $08
0371  8D 01  A0            STA  $A001    Ausgabe des aktuellen Näherungs-
                                         wertes an den D/A-Wandler
0374  A2 20                LDX  #$20     Verzögerungsschleife für
                                         Komparatoreinstellung
0376  CA           LP9     DEX
0377  E0 00                CPX  #$00
0379  10 FB                BPL  LP9
037B  AD 00  A0            LDA  $A000    Lies Komparatorausgang
037E  29 01                AND  #$01     Isoliere Bit 0
0380  C9 01                CMP  #$01
0382  F0 05                BEQ  SHFBIT   Komparatorausgang = 1 bedeutet
                                         Ausgang des D/A-Wandlers noch
                                         zu klein, behalte aktuelle Nähe-
                                         rung bei und schiebe Bit; andern-
                                         falls Ausgang zu groß, subtrahiere
                                         aktuelles Bit vom Näherungs-
                                         wert, dann schiebe Bit.
0384  98                   TYA
0385  45 08                EOR  $08
0387  85 08                STA  $08
0389  98           SHFBIT  TYA
038A  4A                   LSR  A        Schiebe (Y) um 1 Bit nach rechts
                                         für nächsten Näherungsschritt.
038B  A8                   TAY
038C  C9 00                CMP  #$00
038E  F0 08                BEQ  ECHO     (Y) = 0 bedeutet Näherung
                                         beendet, Tonausgabe
0390  18                   CLC
0391  65 08                ADC  $08      Nächster Näherungswert =
                                         alte Näherung plus nächstes Bit
0393  85 08                STA  $08
0395  4C 6F  03            JMP  NXTBIT
0398  A0 F0        ECHO    LDY  #$F0     Verzögerungskonstante für
                                         alle Frequenzen
039A  A5 08                LDA  $08
039C  4A                   LSR  A
039D  85 04                STA  $04
039F  A9 80                LDA  #$80
03A1  05 04                ORA  $04      Berechne entsprechende Fre-
                                         quenzkonstante und speichere
                                         sie nach $0004
03A3  85 04                STA  $04
03A5  20 30  02    SPKR    JSR  TON      Rufe TON zur Aktivierung
                                         des Lautsprechers auf
03A8  88                   DEY
03A9  C0 00                CPY  #$00
03AB  30 F8                BMI  SPKR
03AD  4C 6A  03            JMP  START    Wiederhole Näherungsschleife
```

Bild 5.51: (Fortsetzung)

Wir wollen das Programm nun untersuchen und uns dann Verbesserungen überlegen. Bild 5.51 zeigt das Programm. Die ersten vier Befehle legen die Tore A (Ausgabetor für den D/A-Wandler) und B (Eingabekanal für Komparatorausgang) des 6522#1 fest:

```
ADC      LDA  #$FF
         STA  $A003     DDRA 1 = $FF: Ausgabetor
         LDA  #$00
         STA  $A002     DDRB 1 = $00: Eingabetor
```

Mit den nächsten beiden Befehlen wird hexadezimal „80“ in das Y-Register geladen. Es fungiert als Zeigerregister und wird also mit dem Startwert „10000000“ binär initialisiert.

```
START    LDA  #$80
         TAY
```

Der Speicher $0008 wurde dafür reserviert, den aktuellen Näherungswert zu speichern. Er wird mit „10000000“ binär initialisiert:

```
         STA  $08
```

Nun beginnt die eigentliche Näherungsschleife. Der aktuelle binäre Näherungswert wird aus dem Speicher $0008 gelesen und an den D/A-Wandler ausgegeben:

```
NXTBIT   LDA  $08
         STA  $A001
```

Dann wird eine Verzögerung eingeführt, die dem Komparator ermöglicht, sich auf den richtigen Ausgangswert einzustellen:

```
         LDX  #$20
LP9      DEX
         CPX  #$00
         BPL  LP9
```

Der Ausgang des Komparators wird gelesen:

```
         LDA  $A000     LIES
                        KOMPARATORAUSGANG
```

Bit 0 des IORB wird isoliert und getestet:

```
         AND  #$01      ISOLIERE BIT 0
         CMP  #$01
         BEQ  SHFBIT
```

```
ECHO        LDY  #$F0
            LDA  $08
            LSR  A
            STA  $04
            LDA  #$80
            ORA  $04
            STA  $04
SPK         JSR  TON        AKTIVIERE
                            LAUTSPRECHER
```

Als nächstes wird die Routine TON aufgerufen, die einen Ton der definierten Frequenz erzeugt. Dann wird das Y-Register dekrementiert und auf Null geprüft. Solange der Wert „0“ noch nicht erreicht ist, wird der Ton weiter generiert:

```
            DEY
            CPY  #$00
            BMI  SPKR
            JMP  START
```

Wenn der Lautsprecher den Ton für die vorgegebene Zeit erzeugt hat, kehrt das Programm an den Anfang der Näherungsschleife zurück und fragt erneut den Status des Thermistors ab.

Übungsaufgabe 5.34: *Stellen Sie den gewonnenen Näherungswert hexadezimal auf der Anzeige dar.*

Übungsaufgabe 5.35: *Ist es möglich, alle Befehle „CPY* #$00“ aus dem Programm zu streichen?

Übungsaufgabe 5.36: *Eichen Sie Ihren Thermistor, indem Sie den berechneten Näherungswert bestimmen, der einer gegebenen Temperatur entspricht, die Sie mit einem Thermometer messen. Speichern Sie diese Werte in einer Tabelle, so daß Sie die tatsächliche Temperatur, und nicht den im Speicher stehenden Näherungswert anzeigen können.*

Übungsaufgabe 5.37: *Ändern Sie das Programm so ab, daß der Lautsprecher je nach gemessener Temperatur 1 bis 10 mal ertönt. Bei Raumtemperatur soll er einmal ertönen, bei hoher Temperatur 10 mal. Dies ist eine Möglichkeit, die Meßergebnisse akustisch darzustellen (allerdings mit schlechter Auflösung).*

Falls der Ausgang „1“ ist, ist der Näherungswert noch zu klein, und es muß einfach das nächste Bit gesetzt werden. Ist er „0“, so ist der Näherungswert zu groß, und das aktuelle Bit muß zurückgesetzt werden:

```
                TYA
                EOR  $08
                STA  $08
```

Nachdem so der aktuelle Näherungswert falls nötig korrigiert ist, wird das Zeigerregister nach rechts auf das nächste Bit der Iteration geschoben:

```
SHFBIT          TYA
                LSR  A
```

Wenn das letzte Bit erreicht ist, haben wir die bestmögliche Näherung gewonnen und springen zur Marke ECHO, um den Lautsprecher einzuschalten:

```
                TAY
                CMP  #$00
                BEQ  ECHO
```

Andernfalls setzen wir das nächste Bit der Näherung, und springen an den Anfang der Schleife zurück:

```
                CLC
                ADC  $08
                STA  $08
                JMP  NXTBIT
```

Die Routine ECHO wird in Abhängigkeit vom gemessenen Wert einen Ton erzeugen. In dieser Routine wird das Y-Register bei der Erzeugung der Verzögerung benutzt, während der der Lautsprecher den Ton erzeugt. Es wird hier mit dem Startwert „F0“ hexadezimal geladen. Der Näherungswert wird aus dem Speicher $0008 gelesen und um ein Bit nach rechts geschoben. Das hat zur Folge, daß das letzte Bit des Näherungswertes bei dieser Technik keine Änderung der Tonhöhe bewirkt.

Bit 7 wird zwangsweise auf „1“ gesetzt, so daß der Ton eine gewisse Mindestfrequenz hat, und stets hörbar ist.

Der resultierende Wert wird im Speicher $0004 abgelegt. Dieser Speicher wird, wie bereits beschrieben, dazu verwendet, einen Parameter an die Routine TON zu übergeben:

Übungsaufgabe 5.38: *Wenn Sie die Eichung Ihres Thermistors beendet haben, schließen Sie eine Heizspirale (oder einen Tauchsieder) an die Platine an, und regulieren damit die Temperatur eines Glases Wasser so, daß das Wasser konstant auf einer Temperatur T bleibt (einen Tauchsieder kaufen Sie billig in jedem Haushaltswarengeschäft). Vorsicht: Die meisten Thermistoren sind nicht wasserdicht, so daß man sie eventuell an der Außenwand des Flüssigkeitsbehälters befestigen muß, anstatt sie in die Flüssigkeit direkt zu tauchen. Aber es gibt auch wasserdichte Thermoelemente oder Thermistoren, die die man in flüssigen Medien verwenden kann.*

Übungsaufgabe 5.39: *Als weitere Verbesserung zu Ihrer Einbruchs-Alarmanlage (siehe Programm 5.7) fügen Sie in die Haupt-Kontrollschleife ein Unterprogramm ein, das periodisch die Temperatur mißt. Wenn die Temperatur einen vorgegebenen Wert von beispielsweise 35°C übersteigt, dann soll Alarm ausgelöst werden. Damit haben Sie einen automatischen Brandmelder zugefügt.*

Übungsaufgabe 5.40: *Ein anderer Variationsvorschlag: Sie sollen Ihren Lötkolben gerade so nah an den Thermistor heranbringen, daß dieser eine bestimmte Temperatur – beispielsweise 80°C – annimmt. Ändern Sie Ihr Programm derart ab, daß es eine LED schnell blinken läßt, solange die Temperatur des Thermistors noch wesentlich kleiner als die gewünschte Temperatur ist, und bei Annäherung an den Schwellwert immer langsamer blinkt. Mit einer zweiten LED sollte angezeigt werden, ob die Temperatur über oder unter dem gewünschten Wert liegt.*

Zusammenfassung

In diesem Kapitel haben wir praktische Anwendungen entwickelt, die von einfachen Haushaltssteuerungen bis zu komplexen Industriesteuerungen reichten. Wir haben eine Vielzahl von Ein/Ausgabebausteinen an unsere Mikroprozessorplatine angeschlossen, angefangen mit Schaltern und LED's bis hin zu einem Gleichstrommotor, einem Thermistor und einer Lichtschranke. Die Auswahl der hier vorgestellten Geräte und Techniken sollte Sie in die Lage versetzen, mit der Lösung einer großen Zahl praktischer Steuerungsprobleme zu beginnen. Für eine weiterführende Behandlung spezieller Interfacetechniken verweisen wir auf unser Buch „Mikroprozessor Interface Techniken“. Für die Entwicklung einer soliden Programmiererfahrung sei nochmals dringend das praktische Experimentieren empfohlen.

Im nächsten Kapitel werden wir richtige Peripheriegeräte an unsere 6502-Platine anschließen.



Kapitel 6

Peripheriegeräte

Einleitung

In diesem Kapitel werden wir unseren 6502-Mikrocomputer an richtige Peripheriegeräte anschließen. Die Programme in diesem Teil des Buches sind optimiert, um „elegante" Techniken zur Lösung von Problemen zu demonstrieren. Die Möglichkeiten der einbezogenen Komponenten werden dabei voll ausgeschöpft.

Zuerst werden wir eine normale Matrix-Tastatur mit 16 Tasten anschließen und dabei die Möglichkeiten der Ein/Ausgaberegister geschickt dazu benutzen, die Zahl der für die Erkennung und Anzeige des Zeichens nötigen Befehle minimal zu machen. Als nächstes werden wir uns einen billigen Lochstreifenleser selbst bauen. Bei diesem Gerät kann der Lochstreifen einfach von Hand durch den Leser gezogen werden und wird vom Mikrocomputer korrekt gelesen werden. Zuletzt werden wir noch zeigen, wie einfach es ist, einen Mikroprinter (oder eine ASCII-Tastatur) an den Mikrocomputer anzuschließen. An diesem Punkt angelangt, sollte der Leser soviel Vertrauen in seine Fähigkeiten erlangt haben, daß er sich an die Lösung der meisten Standardprobleme heranwagen kann, die bei wirklichen Anwendungen auftreten.

Die hier vorgestellten Anwendungen sind einfach zu realisieren und sehr nützlich. Alle Programme können mit meist kleinen Abwandlungen direkt auf allen in Kapitel 3 vorgestellten Geräten angewendet werden.

Alle vorgestellten Programme sind kurz, und werden Ihren Wissensstand wertvoll erweitern, auch wenn Sie nicht planen, das betreffende Peripheriegerät anzuschließen. Jedem Leser wird ein sehr sorgfältiges Studium dieses Kapitels empfohlen.

Tastatur

Als erstes werden wir eine externe Matrix-Tastatur mit 16 Tasten (eine sogenannte hexadezimale Tastatur) anschließen, und die gedrückte Taste erkennen. In Bild 6.1 ist zu sehen, wie die Tastatur an die 8 Bit des IORA eines 6522 angeschlossen wird. Die Bits 0 bis 3 sind mit den Zeilen verbunden, während die Bits 4 bis 7 an die Spalten angeschlossen sind. In den Bildern 6.2 bis 6.4 ist die Taste gedrückt, die Zeile 2 und Spalte 7 miteinander verbindet.

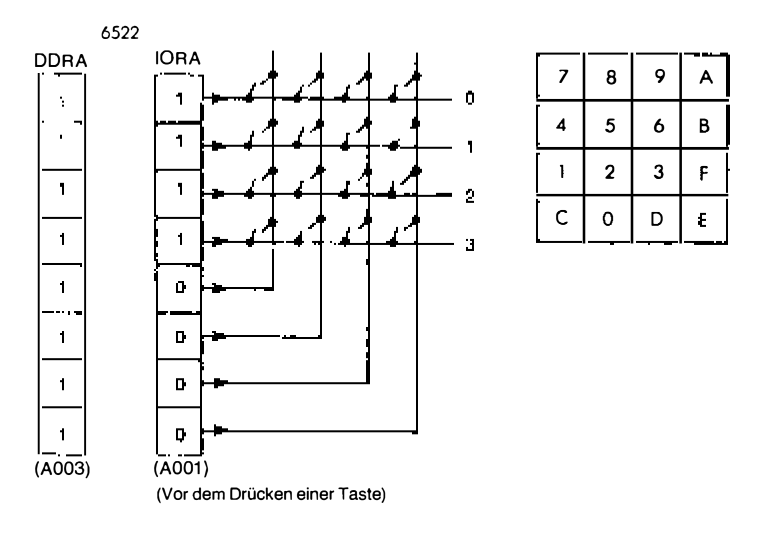


Bild 6.1: Anschluß der Tastatur

Im DDRA wird das IORA als Ausgabetor vereinbart. In diesem Programm nutzen wir eine Besonderheit des IORA der 6522. Das Tor IORA ist ein echtes bidirektionales Register. Wir setzen alle Zeilen auf „1“ und alle Spalten auf „0“. Wird eine Taste gedrückt, so wird die entsprechende Zeile durch diejenige Spalte geerdet, mit der sie über den Tastenkontakt verbunden ist. Beim Zurücklesen des Inhalts des IORA wird die „0“ in der entsprechenden Zeile in das Register geschrieben. In unserem Beispiel wird der resultierende Wert des IORA beim Einlesen binär „00001011“ oder hexadezimal „0B“ sein (Bilder 6.2 und 6.3). Unter Anwendung einer „line-reversal“-Technik (Einzelheiten hierzu entnehmen Sie den Büchern „Chips and Systems“ oder „Mikroprozessor Interface Techniken“. Schreiben wir „11111011“ binär oder „FB“ hexadezimal ins IORA. Da Zeile Nummer 2 auf „0“ liegt (geerdet), wird sie auch Spalte 7 erden. Wenn wir nun den Inhalt des IORA wieder einlesen, finden wir den Wert „01111011“ binär oder „7B“ hexadezimal. Bei jeder Bitposi-

tion, wo im IORA eine „0“ steht, wurden die entsprechende Zeile und Spalte miteinander verbunden. Diese Technik erkennt nicht nur, welche Taste gedrückt wurde, sondern erkennt auch Fehler, wie beispielsweise das gleichzeitige Drücken zweier Tasten. Falls zu einem Zeitpunkt mehr als eine Taste gedrückt ist, dann wird mehr als eine „0“ pro Halbbyte (Gruppe aus 4 Bit) im IORA auftauchen.

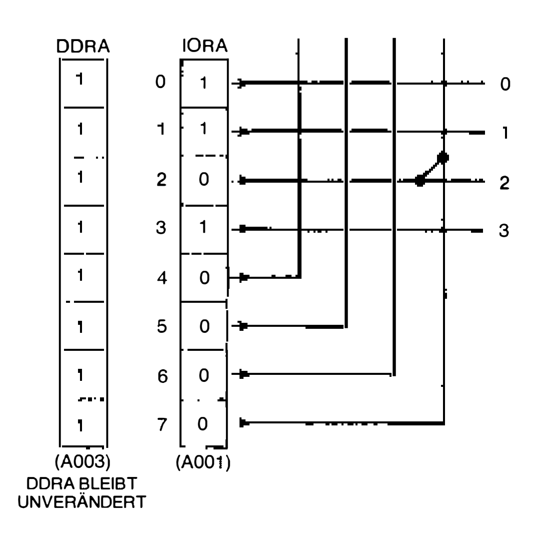


Bild 6.2: Zweiter Schritt — Lesen des IORA nach Schließen des Tastenkontaktes

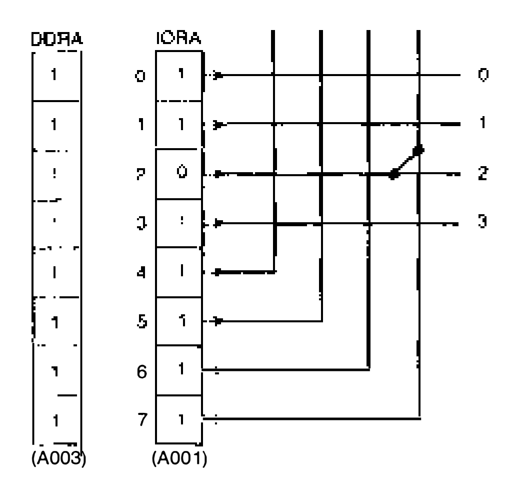


Bild 6.3: Dritter Schritt — Schreiben in IORA

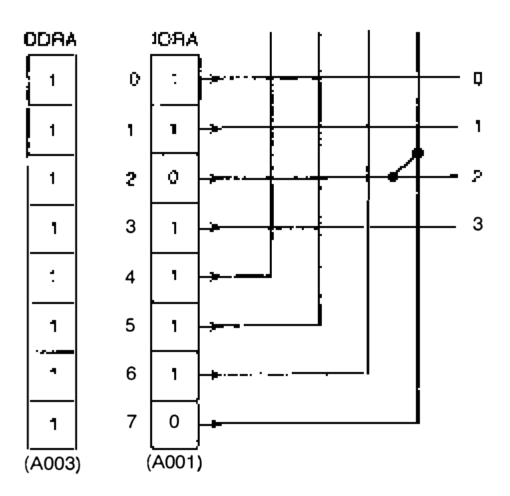


Bild 6.4: Vierter Schritt – IORA wieder einlesen

Um das Zeichen zu erkennen, das der jeweils gedrückten Taste entspricht (ein hexadezimales Zeichen zwischen „0“ und „F“), legen wir einfach eine Tabelle an, die für jedes zulässige Bitmuster des IORA das entsprechende ASCII-Zeichen angibt.

So haben wir zum Beispiel gerade festgestellt, daß beim Drücken der Taste „B“ im IORA „7B“ hexadezimal erscheint. Als Übung empfehlen wir Ihnen, die Kodes für die restlichen 15 Zeichen zu errechnen. In Bild 6.5 finden Sie die entsprechende Tabelle.

Falls ein unerlaubter Kode auftritt, wird er nicht berücksichtigt, und die Tastatur wird erneut abgefragt.

Schließlich kann das Zeichen, sobald sein ASCII-Code festgestellt wurde, angezeigt werden. Als Beispiel wird zur Anzeige des Zeichens die Display-Routine, die Teil des SYM-Monitors ist, verwendet. Am Ende dieses Abschnitts werden wir Änderungsmöglichkeiten vorschlagen, wie man das Zeichen anders ausgeben kann.

ZEICHEN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
IORA	DE	ED	DD	BD	EB	DB	BB	E7	D7	B7	77	7B	EE	BE	7E	7D
ASCII	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Bild 6.5: Kodetabelle für die Zeichen der Hexadezimal-Tastatur

Hinweis: Das Programm benutzt 3 Monitorroutinen: SCAND, HDOUT und ACCESS.

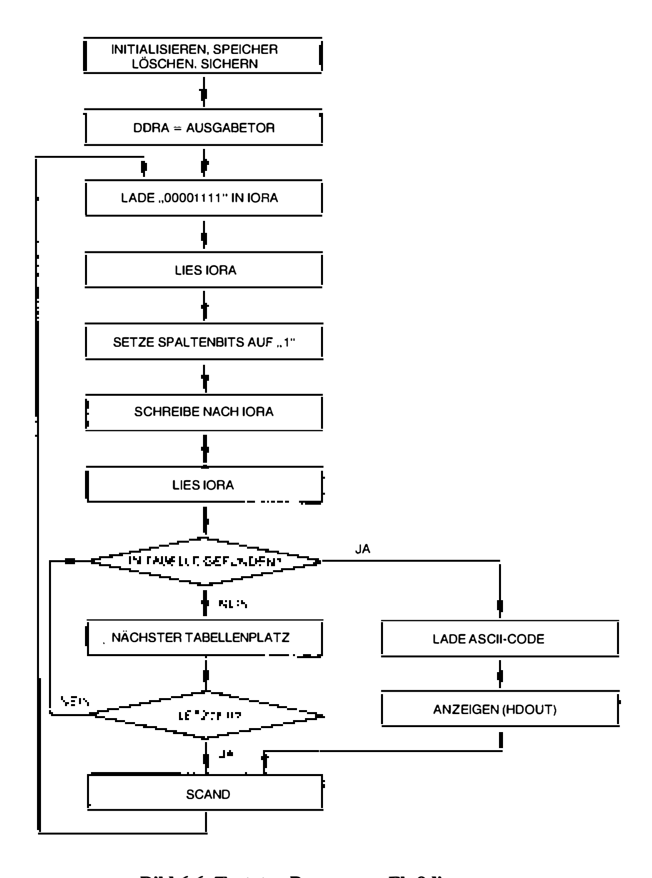


Bild 6.6: Tastatur-Programm, Flußdiagramm

Bild 6.6 zeigt das Flußdiagramm für dieses Programm.

Nach dem Initialisieren des Programms wird „0F" in das IORA geladen. Ohne Änderungen im DDRA(!) wird das IORA dann wieder gelesen. Den gelesenen Wert braucht man wegen der Bidirektionalität des IORA des 6522 nicht in einem 6502-Register oder im RAM zwischenzuspeichern. *Es wird in dem Ein/Ausgabe-Baustein selbst zwischengespeichert und verbleibt dort.* Dann werden die vier Spaltenbits auf „1" gesetzt, und dieses neue Byte wird in das IORA geladen. Nun wird der Inhalt des IORA wieder eingelesen, so daß man das endgültige Bitmuster erhält. Das Bitmuster in dem Ein/Ausgabe-Register IORA wird dann mit allen

möglichen Werten in der IORA/ASCII-Tabelle nach Bild 6.5 verglichen. Falls der Inhalt des IORA mit dem betrachteten Tabellenwert nicht übereinstimmt, wird der nächste Tabellenplatz untersucht. Wird kein übereinstimmender Kode gefunden, so wird ein Rücksprung an den Schleifenanfang durchgeführt.

Das Programm ist in Bild 6.7 aufgelistet.

```
0000  20  86  8B  INIT    JSR   ACCESS
   3  A9  FF              LDA   #$FF
   5  8D  03  A0          STA   DDRA    DDRA is PAD
   8  A2  OF      START   LDX   #$0F
   A  8E  01  A0          STX   IORA     IORA is PA
   D  AD  01  A0          LDA   IORA     IORA is PA
0010  09  F0              ORA   #$F0
   2  8D  01  A0          STA   IORA     IORA is PA
   5  AD  01  A0          LDA   IORA     IORA is PA
   8  D5  30      LOOP    CMP   TAB, X
   A  F0  05              BEQ   DISPL
   C  CA                  DEX
   D  10  F9              BPL   LOOP
   F  30  05              BMI   SCAN
0021  B5  40      DISPL   LDA   ASCT, X
   3  20  00  89          JSR   HDOUT
   6  20  06  89  SCAN    JSR   SCAND
   9  4C  08  00          JMP   START
0030  E7  D7  B7  77 TAB  BYTE  $E7, $D7, $B7, $77, $EB, $DB,
      EB  DB  BB  7B            $BB, $7B, $ED, $DD, $BD,
      ED  DD  BD  7D            $7D, $EE, $DE, $BE, $7E
      EE  DE  BE  7E
0040  37  38  39  41 ASCT BYTE  '7, '8, '9, 'A, '4, '5, '6,
      34  35  36  42            'B, '1, '2, '3, F, 'C, '0,
      31  32  33  46            'D, 'E
      43  30  44  45
```

Bild 6.7: Tastatur-Programm (Programm 6.1)

Im Programmteil INIT (Initialisierung) wird – im Beispiel des SYM durch Aufruf der Routine ACCESS – die Speichersicherung abgeschaltet, und dann das Datenrichtungsregister des Tors A auf Ausgabe geschaltet:

```
INIT          JSR  ACCESS    (nur SYM!)
              LDA  #$FF      „11111111“ = AUSGABETOR
              STA  DDRA
```

Dann wird „00001111“ binär in das Datenregister IORA geladen:

```
START         LDX  #$0F      „00001111“
              STX  IORA
```

Es wird sofort wirder ausgelesen und alle Spaltenbits werden durch eine ODER-Verknüpfung mit „11110000“ binär auf „1“ gesetzt:

```
              LDA  IORA
              ORA  #$F0      „11110000“
```

Das resultierende Byte wird wieder in das Datenregister IORA geladen:

```
              STA  IORA
```

Es wird sofort wieder ausgelesen und enthält nun das endgültige Bitmuster, das wir verwenden, um festzustellen, welche Taste gedrückt worden war:

```
              LDA  IORA
```

Der im Akkumulator stehende Kode wird nun nacheinander mit allen Werten in der Tabelle verglichen. Jedes Mal, wenn wir mit einer Tabelle arbeiten, wenden wir normalerweise eine *indizierte Adressierung* an, um auf die Tabellenelemente nacheinander zuzugreifen. Der Startwert des Indexregisters ist „0F“ hexadezimal oder „15“ dezimal. Die letzte Eintragung in der Tabelle wird daher zuerst verglichen (siehe Bild 6.8). Dann wird die vorhergehende getestet. Wird Übereinstimmung gefunden, so erfolgt ein Sprung zur Marke DISPL:

```
LOOP          CMP  TAB,X
              BEQ  DISPL
              DEX
              BPL  LOOP
```

Liegt keine Übereinstimmung vor, so wird das Index-Register X dekrementiert und zeigt damit auf die nächste Tabelleneintragung. Es muß auf den Wert „0“ getestet werden: wenn es beim Dekrementieren negativ wird, ist keine existierende Taste gefunden worden, und es erfolgt ein Sprung zur Marke SCAN:

```
              BMI  SCAN
```

Bei der Marke DISPL zeigt das X-Register an, welches Zeichen erkannt worden ist. Es enthält eine Zahl zwischen „0“ und „15“ dezimal. Wir wollen diese Zahl nun in den entsprechenden ASCII-Code umwandeln, den

wir zur Anzeige oder zum Ausdrucken des erkannten Zeichens benötigen:

```
DISPL          LDA  ASCT,X
```

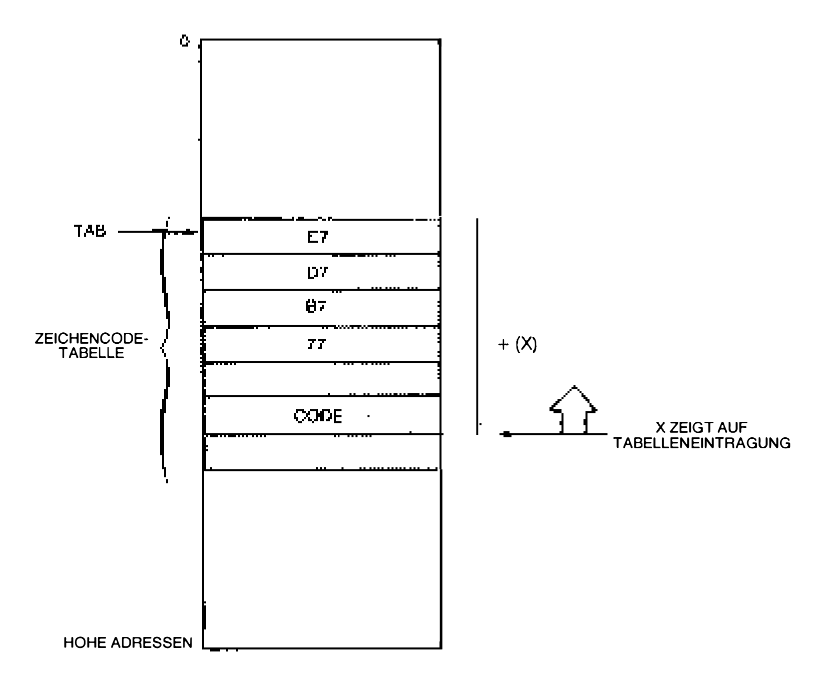


Bild 6.8: Tabellenzugriff durch indizierte Adressierung

An dieser Stelle wird der Akkumulator mit demjenigen ASCII-Kode geladen, der dem durch den Inhalt des X-Registers festgelegten Zeichens entspricht. Wiederum wird für den Zugriff auf diese sequentiell angeordneten Daten die Technik der *indizierten Adressierung* verwendet (siehe Bild 6.9). Dann wird das (SYM-)Unterprogramm HDOUT aufgerufen und das Zeichen (mit der SCAND-Routine des SYM) angezeigt. Anschließend wird die Tastatur-Abfrage wiederholt:

```
               JSR  HDOUT
SCAN           JSR  SCAND
               JMP  START
```

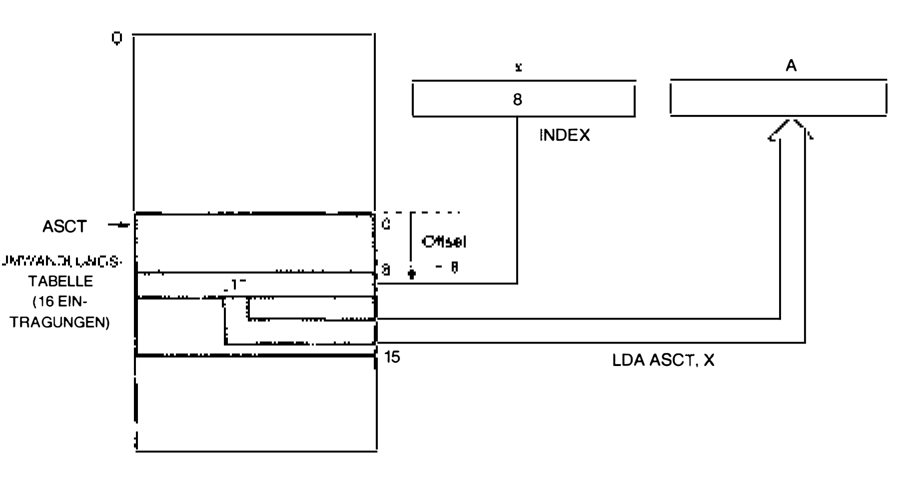


Bild 6.9: Umwandlung des IORA-Kodes in ASCII

Das Programm benutzt zwei Tabellen mit Konstanten. Die erste davon heißt „TAB". Diese Tabelle enthält die Liste der zulässigen Bitmuster des Datenregisters IORA. Durch den Wert des Index-Registers X beim Lesen einer dieser Eintragungen wird jeweils eine entsprechende Taste festgelegt. Die zweite Tabelle heißt „ASCT". Sie enthält die ASCII-Codes für alle 16 Tasten.

Diese beiden Tabellen erscheinen am Ende des Programms in Bild 6.7. Beachten Sie, *daß das X-Register nicht das tatsächliche hexadezimale Zeichen, das der gedrückten Taste entspricht, enthalten muß.* Solange nur beide Tabellen in derselben Reihenfolge angeordnet sind, wird der korrekte ASCII-Code für jedes zulässige binäre Bitmuster, das in der Tabelle „TAB" gefunden wird, bestimmt. Aus diesem Grund mußten die beiden Tabellen in dem Programm nicht in der hexadezimalen Reihenfolge erstellt werden.

Übungsaufgabe 6.1: *Ordnen Sie die beiden Tabellen TAB und ASCT in Bild 6.7 so um, daß der Wert des Index-Registers X stets gleich dem hexadezimalen Wert der gedrückten Taste ist.*

Übungsaufgabe 6.2: *Benennen Sie, als Alternative zu der Methode in Aufgabe 6.1 die Tasten auf der Tastatur ohne Änderung der Tabellen TAB und ASCT so um, daß der Wert des Index-Registers X stets der gedrückten Taste entspricht.*

Wir wollen nun einige Änderungen vorschlagen, so daß das erkannte Zeichen der Außenwelt auch anders angezeigt werden kann:

Übungsaufgabe 6.3: *Beim Drücken der Taste „1" soll der Lautsprecher einmal ertönen, bei der Taste „2" zweimal, etc.*

Übungsaufgabe 6.4: *Ändern Sie unter Verwendung des in Kapitel 4 entwickelten Morseprogramms (Programm 4.3) das Programm so ab, daß nach jedem Tastendruck das jeweilige Morsezeichen ertönt.*

Übungsaufgabe 6.5: *Ändern Sie das obige Programm so ab, daß es nach jedem Tastendruck eine bestimmte Note spielt. Einer Taste sollte dabei keine Note, d. h. eine Pause, zugeordnet sein. Mit zwei anderen Tasten soll man die Tonlänge (1, 2 oder 4 Schläge) bestimmen können.*

Übungsaufgabe 6.6: *Schreiben Sie ein speicherndes Musikprogramm. Zuerst spielen Sie eine Melodie, indem Sie die Tasten in der gewünschten Reihenfolge drücken. Die ersten 50 Noten der Melodie (oder irgendeine andere Anzahl) sollten in den RAM's gespeichert werden. Beim Drücken einer bestimmten Taste sollte das Programm die gespeicherte Melodie vom Anfang an wiedergeben.*

Lochstreifenleser oder ASCII-Tastatur

Das Anschließen einer dekodierten (ASCII) Tastatur oder eines Lochstreifenlesers bedingt praktisch die gleiche Technik. Das Hardwareinterface umfaßt 8 Datenbits (die 7 Bit ASCII-Code plus ein Paritätsbit) und ein separates Status-Bit, mit dem angezeigt wird, daß ein Zeichen ansteht. Wir weren ein Interface für einen „Eigenbau"-Lochstreifenleser in vereinfachter Form vorstellen. Das Programm für eine dekodierte Tastatur wäre praktisch identisch.

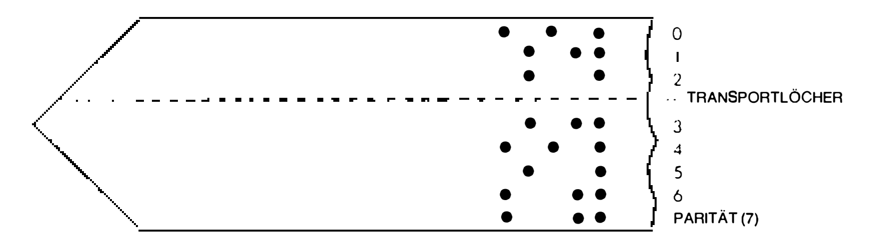
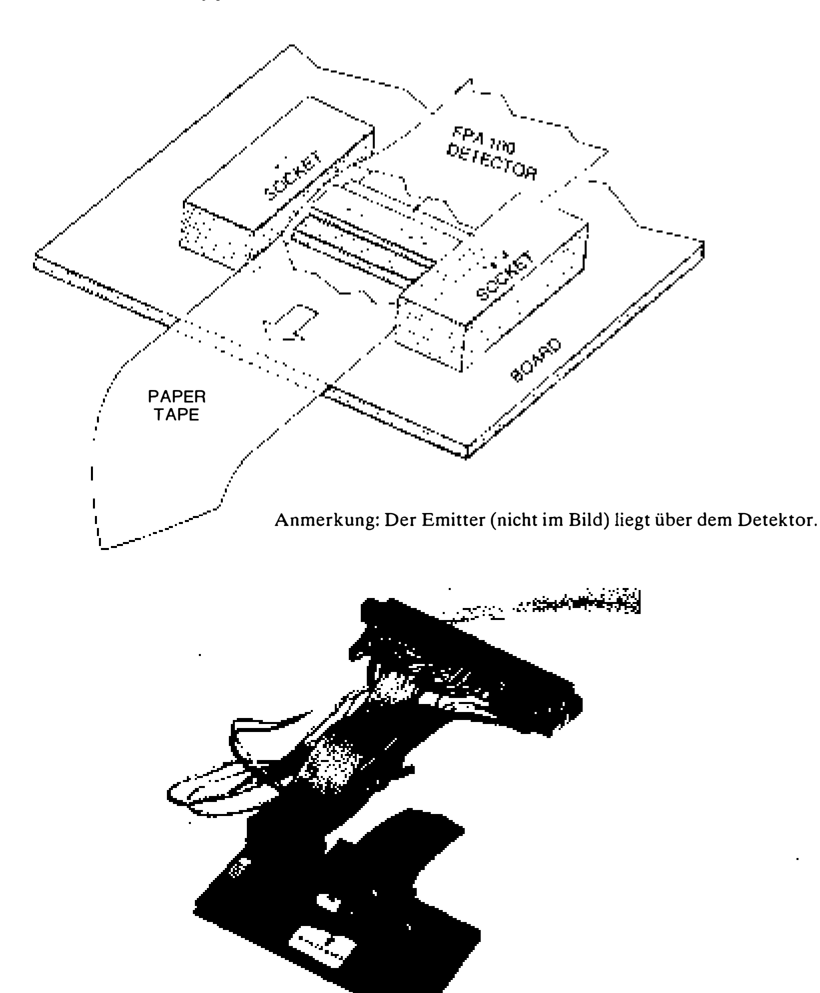


Bild 6.10: Ein 8-Bit Lochstreifen

Ursprünglich wurden Lochstreifen zur Speicherung von Programmen in einer zuverlässigen und ökonomischen Form eingesetzt. Jedes Zeichen wird auf dem Lochstreifen durch eine Reihe eingestanzter Löcher dargestellt (siehe Bild 6.10). In ein zusätzliches Loch, das kleiner als die anderen ist, greift ein Zahnrad und transportiert den Lochstreifen weiter. Gleichzeitig wird der Lochstreifen dadurch richtig positioniert. Die anderen 8 Löcher (es gibt auch Kodes mit weniger Löchern) dienen zur Verschlüsselung des eigentlichen Zeichens im ASCII-Format. Mit dem Zahn-

rad wird der Lochstreifen Loch für Loch weitertransportiert, wobei der eingestanzte Lochcode von einer Leseeinrichtung jeweils gelesen werden muß. Wir werden in unserem Fall hierfür einen Satz Fotoemitter und Fotodetektoren des Typs FPA100 verwenden.



Anmerkung: Der Emitter (nicht im Bild) liegt über dem Detektor.

Bild 6.11: Hardware des Lochstreifenlesers

Der FPA100-Emitter sitzt zuoberst auf der kleinen Platine. Der Lochstreifenleser wird an die 6502-Platine mit einem Flachbandkabel an den A-Stecker angeschlossen.

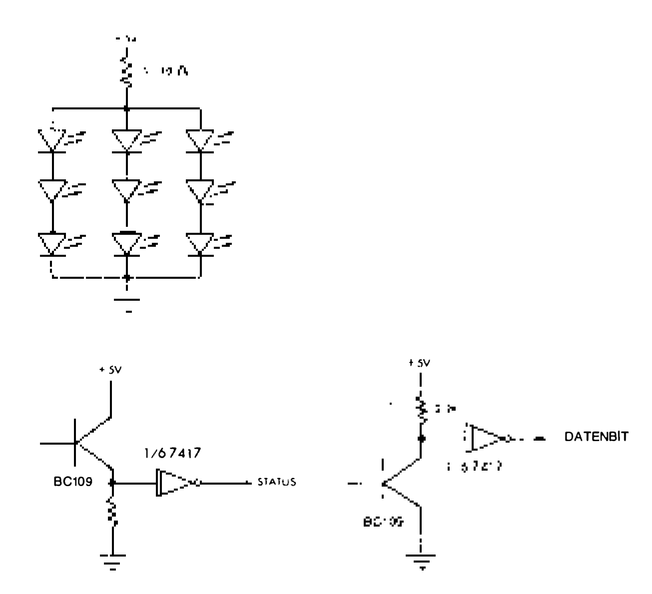


Bild 6.12: Lochstreifenleser, Anschlußdetails

Die lichtemittierenden Dioden emittieren kontinuierlich Licht. Wenn ein Loch vor die LED zu liegen kommt, geht das Licht hindurch, und wird von dem Fotodetektor auf der anderen Seite empfangen. Dies entspricht einer „1". Wird kein Licht durchgelassen, bedeutet das eine „0". Beach-

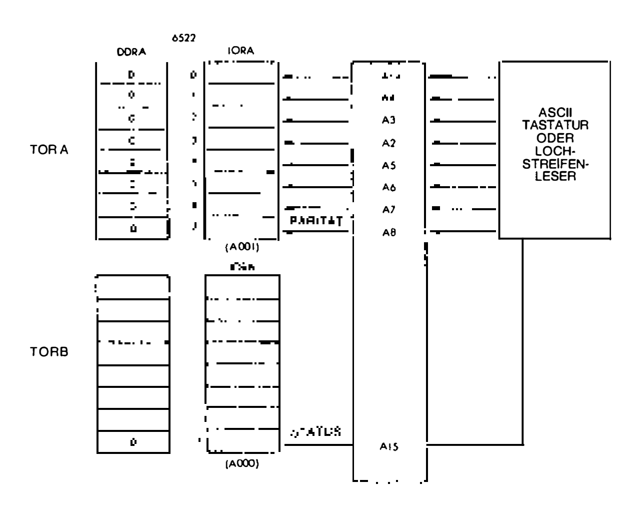


Bild 6.13: Lochstreifenleser, Interface

ten Sie, daß die Intensität der LED's sorgfältig eingestellt sein muß, damit das Licht nicht schon ohne ein Loch durch den Lochstreifen schimmert (praktische Hinweise hierzu geben wir später). Dieser sehr preiswerte und einfache Lochstreifenleser kann von Hand betrieben werden, indem man einfach den Lochstreifen zwischen Emitter und Detektor hindurchzieht. Wie wir sehen werden, synchronisiert sich das Programm selbst mit Hilfe der Transportlöcher, in die normalerweise das Zahnrad greift. Das Hardware-Diagramm erscheint in Bild 6.11. Die Anschlußdetails für die lichtemittierenden Dioden, die Lochdetektor- und die Datendetektor-Schaltung zeigt Bild 6.12. Das Mikrocomputer Interface ist in Bild 6.13 gezeigt. Das IORA des 6522#1 wird als Eingabetor für die Datenbits benutzt. Das IORB desselben 6522 wird dazu verwendet, mit Bit 7 das Statusbit zu lesen. Steht kein zweites Tor zur Verfügung (z. B. CBM), so wird man das Statussignal z. B. über CA1 einlesen.

Die Signale werden mit Schmitt-Triggern (7414) aufbereitet. Die beiden Sockel für die 7414-IC's werden gleichzeitig als Führung für den Lochstreifen verwendet. Das Signal, das beim Nachweis eines *Transportloches* auftritt, ist eine „0", das beim Nachweis eines *Datenloches* eine „1".

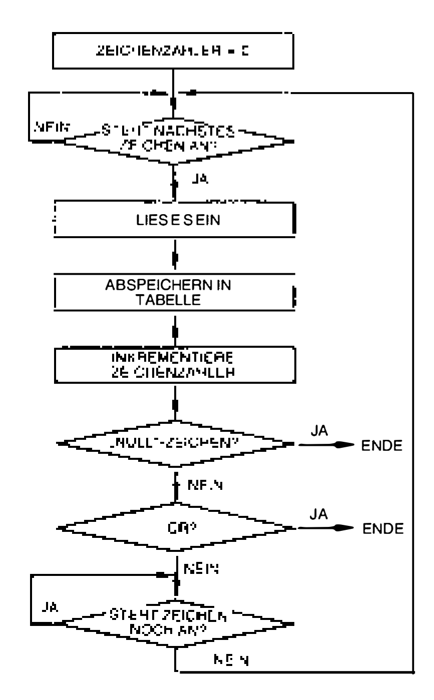


Bild 6.14: Programm Lochstreifenleser, Flußdiagramm

Beachten Sie, daß bei diesem einfachen Interface zum Treiben aller LED's nur ein einziger Widerstand benutzt wird. In der Praxis sollte für jede einzelne LED ein separater Vorwiderstand verwendet werden. Der Wert dieses Wiederstandes muß sehr sorgfältig so gewählt werden, daß gerade so viel Licht durch ein Loch gelangt, daß es vom gegenüberliegenden Detektor nachgewiesen werden kann. Andernfalls kann es passieren, daß *nur Einsen* („11111111") am Ausgang erscheinen, da das Licht durch einen normalen (ziemlich transparenten) Lochstreifen etwas hindurchschimmert. Falls Sie durch die Wahl des Widerstandes dieses Problem nicht beseitigen können, läßt sich durch Verwendung *schwarzen*, oder doch wenigstens weniger transparenten Lochstreifens Abhilfe schaffen.

Bild 6.14 zeigt das Flußdiagramm. Zum Abzählen der Anzahl eingelesener Zeichen wird ein Zeichenzähler verwendet. Das Programm verbleibt in einer Warteschleife, bis das nächste Zeichen ansteht. Dies wird dadurch erkannt, daß ein Transportloch über den entsprechenden Detektor zu liegen kommt. Sobald das Statussignal das Anstehen eines Zeichens meldet, sollte dieses möglichst schnell eingelesen werden. Es wird eingelesen und in einer sequentiellen Tabelle im Speicher abgelegt. Anschließend wird der Zeichenzähler inkrementiert.

Üblicherweise wird das Einlesen entweder durch ein „NULL"-Zeichen (überhaupt kein Loch gestanzt) oder durch ein explizites „Wagenrücklauf"-Zeichen (CR, engl. carriage return) beendet. Das Programm testet daher das eingelesene Zeichen auf „NULL" oder „CR" und endet, falls eines dieser beiden Zeichen gefunden wird. Liegt keines dieser Zeichen vor, so kann es an den Schleifenanfang zurückspringen. Vorher muß es jedoch noch warten, bis die Stausinformation zurückgesetzt worden ist. Sobald dieses „Zeichen vorhanden"-Signal verschwunden ist, springt das Programm an den Schleifenanfang zurück und wartet auf das Anstehen des nächsten Zeichens.

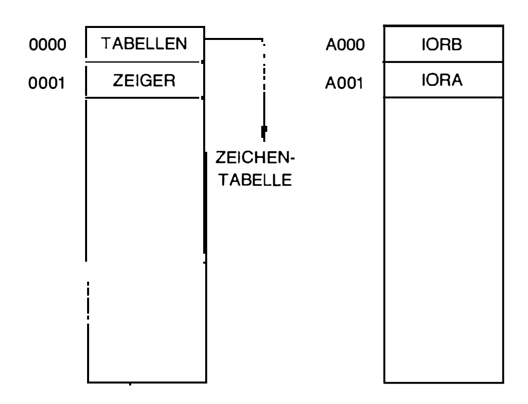


Bild 6.15: Lochstreifenleser, Speicherbelegung

Die zum Programm gehörende Speicherbelegung zeigt Bild 6.15, das Programm selbst Bild 6.16.

```
0002  A0 00      KBPT   LDY  #$00
0004  2C 00  A0  TS     BIT  IORB      Teste Statusbit
0007  30 FB             BMI  TS
0009  AD 01  A0         LDA  IORA      Lade Datenwort
000C  91 00             STA  ($00),Y
000E  C8                INY
000F  C9 00             CMP  #$00
0011  F0 0B             BEQ  RET
0013  C9 8D             CMP  #$8D
0015  F0 07             BEQ  RET
0017  2C 00  A0  TE     BIT  IORB      Teste Statusbit
001A  10 FB             BPL  TE
001C  30 E6             BMI  TS
001E  60         RET    RTS
```

Bild 6.16: Lochstreifenlese/ASCII-Tastatur-Programm (Programm 6.2)

Das Programm setzt voraus, daß DDRA und DDRB schon mit den richtigen Werten initialisiert worden sind. Andernfalls müssen am Programmanfang entsprechende Befehle eingefügt werden. Das Y-Register wird als Zeichenzähler verwendet und mit „0“ initialisiert:

```
KBPT        LDY  #$00
```

Als nächstes wird der Wert des Statusregisters getestet, um festzustellen, ob ein Zeichen ansteht. Um es leicht erkennen zu können, wurde es an Bit 7 des IORB gelegt:

```
TS          BIT  IORB
            BMI  TS
```

Bit 7 ist für den Anschluß eines Statussignals ein bevorzugtes Bit, da es mit einem einzigen Befehl abgefragt werden kann: Bit 7 ist das „Vorzeichen“-Bit. Das Statussignal setzt im Statusregister das N-Flag, das direkt auf „positiv“ und „negativ“ (entsprechend „0“ und „1“) getestet werden kann. In diesem Fall wird es durch den Befehl BMI (springe bei minus,

engl. branch on minus) abgefragt. Solange das Signal „1“ ist, liegt kein Zeichen an. Wenn es „0“ wird, liegt ein Zeichen an. Der Akkumulator kann dann mit dem Datenwort, das an den Datenleitungen anliegt, geladen werden:

LDA IORA LADE DATENWORT

Das vom Lochstreifenleser übertragene 8-Bit Datenwort muß dann in einem geeigneten Speicher abgelegt werden. Wir wollen dabei annehmen, daß die Anfangsadresse des Puffers in den Speichern „00,01“ abgespeichert worden ist. Wir werden für den Zugriff auf das erste Element der Tabelle eine *indirekte Adressierung* anwenden. Zusätzlich wird die Adressierung durch den Wert des Y-Registers *indiziert*, um nacheinander auf alle Tabellenelemente zugreifen zu können. Der entsprechende Befehl lautet:

STA ($00),Y

Diesen Indirekt indizierten Befehl wollen wir uns ein weniger näher ansehen. *Indirekt* bedeutet: „Gehe zum Speicher $00 und verwende seinen und den im nachfolgenden Speicher stehenden Inhalt als Adresse“ (erster Schritt in Bild 6.17).

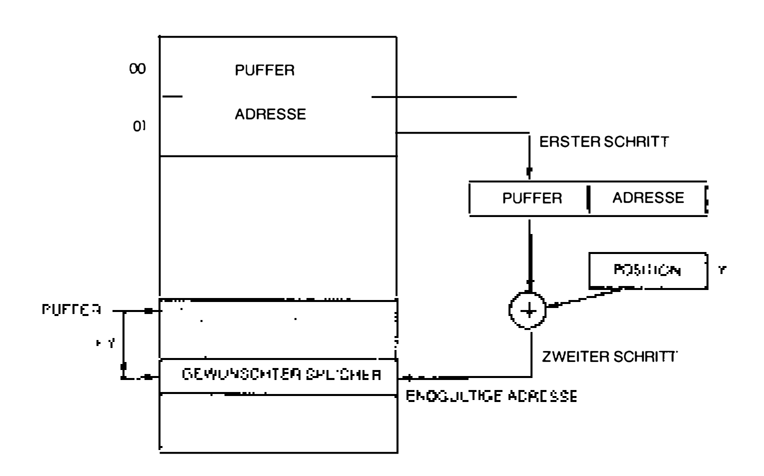


Bild 6.17: Indirekt indizierter Zugriff: STA ($00),Y

Dann wird das Y-Register als Index verwendet: sein Inhalt wird zur Basisadresse addiert und ergibt die endgültige Adresse (zweiter Schritt in

Bild 6.17). Der Inhalt des Y-Registers gibt die Verschiebung innerhalb des Pufferspeichers an, d. h. die relative Adresse in Bezug auf den Tabellenanfang.

Nun wird der Speicherzähler inkrementiert und zeigt damit auf den nächsten Speicherplatz in der Tabelle, in den später das nächste Zeichen geschrieben wird:

```
                INY
```

Das Zeichen im Akkumulator muß nun auf „NULL" oder „Wagenrücklauf (CR)" getestet werden, um festzustellen, ob das Ende der Zeile erreicht ist. Dies erledigen die nächsten vier Befehle:

```
                CMP  #$00      NULL ?
                BEQ  RET       FALLS JA, DANN ENDE
                CMP  #$8D      CR ?
                BEQ  RET       FALLS JA, DANN ENDE
```

Zum Schluß müssen wir warten, bis das Statussignal, das das Anliegen eines Zeichens signalisiert, wieder zurückgesetzt ist (siehe Flußdiagramm Bild 6.14), und können erst dann den Abfragezyklus wiederholen. Andernfalls würden wir dasselbe Zeichen mehrmals lesen. Das wird mit den nächsten drei Befehlen durchgeführt:

```
TE              BIT  IORB      TESTE STATUSSIGNAL
                BPL  TE
                BMI  TS
```

Das komplette Unterprogramm endet wie üblich mit dem Befehl Return:

```
RET             RTS
```

Übungsaufgabe 6.7: *Lassen Sie zusätzlich zum Abspeichern des Zeichens in der Tabelle über den Lautsprecher den Morse-Kode des eingelesenen Zeichens ausgeben. Achten Sie darauf, das Morsezeichen schnell genug zu erzeugen, damit Sie bei der Eingabe keine Zeichen verlieren. Oder aber Sie ziehen den Lochstreifen so langsam durch den Leser, daß zwischen zwei aufeinanderfolgenden Zeichen genügend Zeit für das Morsezeichen bleibt. Eine andere Möglichkeit wäre, die Morsezeichen erst am Ende, wenn alle Zeichen eingelesen sind, zu erzeugen. Das ist zwar bestimmt die sicherste Lösung, aber es entgeht Ihnen dabei die Möglichkeit, nachzuprüfen, ob jedes Zeichen tatsächlich richtig eingelesen worden ist!*

Übungsaufgabe 6.8: *Schließen Sie auf der Platine mit dem Lochstreifenleser acht LED's an, und schalten Sie diese mit der CPU entsprechend dem gelesenen Zeichen ein.*

Übungsaufgabe 6.9: *Lassen Sie ein Alarmsignal ertönen, falls das Paritätsbit falsch ist. (Das Paritätsbit legt – je nachdem, wie es vereinbart ist – fest, ob die Anzahl der gesetzten Bits eines gegebenen Zeichens gerade oder ungerade ist. Das müssen Sie aber nachprüfen!)*

Mikrodrucker

Viele Mikrodrucker verwenden elektrosensitives Papier und drucken 20 Zeichen pro Zeile. Zur Darstellung der Zeichen wird eine Punktmatrix benutzt. Beispiele sind Olivetti (verschiedene Modelle) und Matsushita. Der Drucker selbst benötigt ein kleines Interface, das die entsprechenden Signale an den Druckerkopf schickt, den Papiervorschub und die Druckmechanik steuert. Einmal mit einem solchen Standard-Interface ausgerüstet, *kann der Mikrodrucker an jeden Mikroprozessor angeschlossen werden, der mit einem PIO (also einem programmierbaren Ein/Ausgabe-Tor) ausgerüstet ist.* Einen solchen Drucker werden wir hier benutzen und in diesem Beispiel über einen 6522 und einen 6532 an das 6502-System anschließen. Jedoch ist prinzipiell ein einziges 8-Bit-Tor ausreichend. Falls Sie einen Drucker mit einem anderen Interface benutzen, können sich Unterschiede ergeben. Die Logik des Programms sollte jedoch im wesentlichen erhalten bleiben.

Das Programm druckt jeweils 20 Zeichen pro Zeile. Es erzeugt ein „Startsignal" und übermittelt dann nacheinander 20 Zeichen. Vor der Übermittlung jedes Zeichens wartet das Programm, bis das Druckerinterface ein „Zeichen-Anforderungs-Signal" (request) gibt. Auf dieses Signal hin muß das Programm die Zeichen übermitteln, da sonst fälschlicherwei-

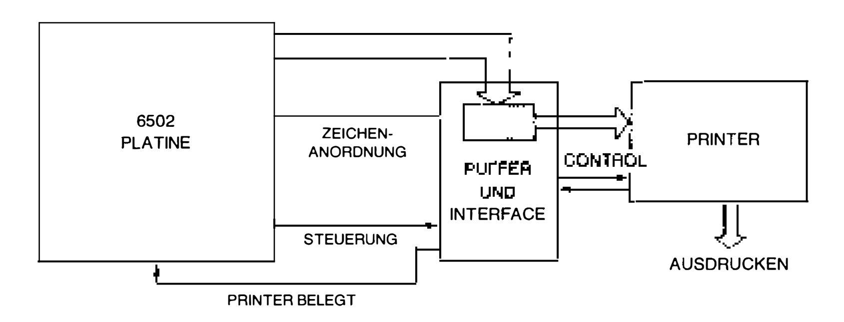


Bild 6.18: Standard-Druckerinterface

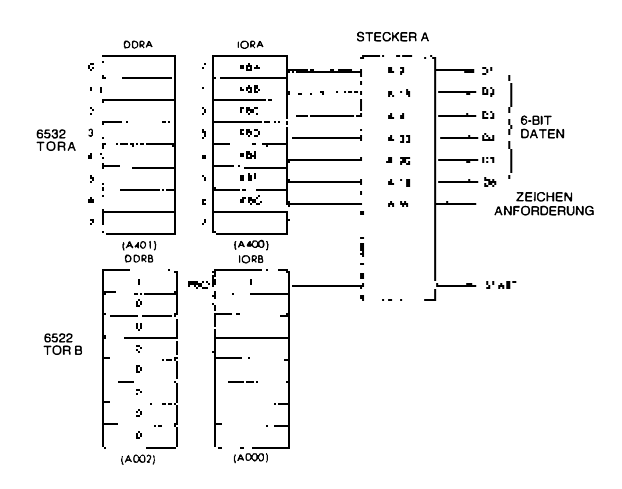


Bild 6.19: Anschluß des Druckers

se die noch im Interface-Puffer befindlichen vorhergehenden Zeichen gedruckt werden. Das Zeichen wird über die 6 Datenleitungen übermittelt. Es wird also eine 6-Bit-Darstellung gewählt (siehe Bild 6.18).

Das Anschlußbild für den Drucker zeigt Bild 6.19. Vom 6532 wird Tor A verwendet, ferner vom 6522 das Bit 0 des Tors B. Das IORA des 6532 stellt die 6 Datenleitungen und empfängt auf Bit 6 das „Zeichen-Anforderungs-Signal", wie auch im Bild 6.19 zu sehen ist. Das Bit 0 des IORB des 6522 wird zur Erzeugung des Startsignals benutzt. Zusätzlich liefert das Druckerinterface normalerweise ein „Drucker belegt"-Signal (engl. „busy"). Es wird hier nicht berücksichtigt. Wir ersetzen es durch eine Software-Verzögerungsroutine von 48 msec. Ein Flußdiagramm für das Programm zeigt Bild 6.20.

Die Datenrichtungsregister der beiden PIO's werden initialisiert und ein Startsignal an den Drucker geschickt. Das Programm prüft dann die „Zeichen-Anforderungs"-Leitung (request). An diesem Punkt wartet das Programm so lange, bis ein Spannungssprung anzeigt, daß ein Zeichen angefordert wird. Aus einem der Speicherbereiche, wo die 20-Zeichen-Zeilen abgelegt sind, wird das nächste Zeichen geholt (siehe Bild 6.21). Dann wird dieses Zeichen an den Drucker geschickt. Nach der Übermittlung des Zeichens wartet das Programm, bis das Zeichen-Anforderungs-Signal zurückgesetzt wird. Es inkrementiert den Zeichenzähler und überprüft, ob dieser den Wert „20" erreicht hat. Falls der Wert „20" noch nicht erreicht ist, muß das nächste Zeichen an den Drucker übermittelt werden, und die Schleife wird erneut durchlaufen. Sobald an den Drucker 20 Zeichen geschickt worden sind, wird ein „space"-Zeichen an den Drucker gegeben, das die Zeile beendet, und einen Papiervorschub sowie einen Wa-

genrücklauf bewirkt (bei anderen Interfaces kann diese Vereinbarung anders getroffen sein). Dann muß eine Verzögerung von 48 msec durchgeführt werden, damit die mechanischen Elemente des Druckers auf die nächste Zeile fahren können. Hierfür wird der interne Zeitgeber des 6532 im Modus „1024-fach" verwendet. Der Faktor 1024 bedeutet, daß jede Einheit des Inhalts des Zeitgeberregisters eine Verzögerung von 1024 µsec, also etwa 1 msec, bewirkt. Das Zeitgeberregister wird mit „30" hexadezimal = „48" dezimal geladen. Das Programmm endet mit dem Ablauf dieser Verzögerung.

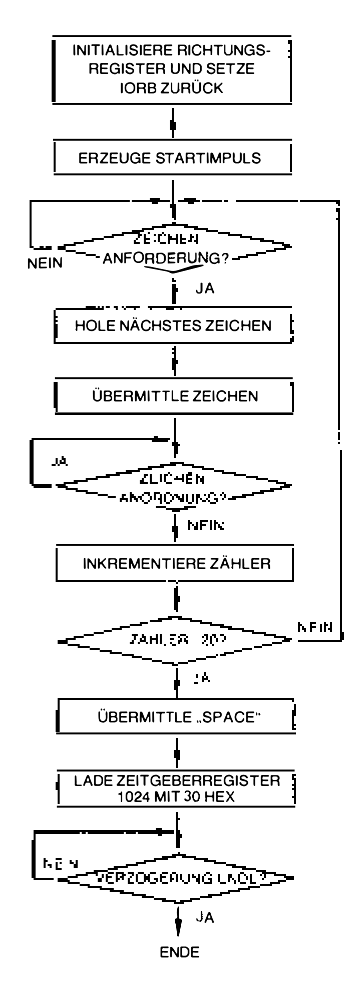


Bild 6.20: Flußdiagramm für Druckerprogramm

Das Programm ist in Bild 6.22 gezeigt. Die zugehörige Speicherbelegung finden Sie in Bild 6.21. Die beiden Speicher „00“ und „01“ enthalten den Zeiger auf die Adresse des ersten Zeichens im Speicher. Um das Programm zu benutzen, sollte der Anwender vor Einschalten des Druckers den Wert „01“ nach „A002“ (DDRB) und „00“ nach „A000“ (IORB) schreiben. Bild 6.19 zeigt, welche Speicher von den Ein/Ausgabe-Bausteinen verwendet werden. Im folgenden wollen wir das Programm genauer untersuchen.

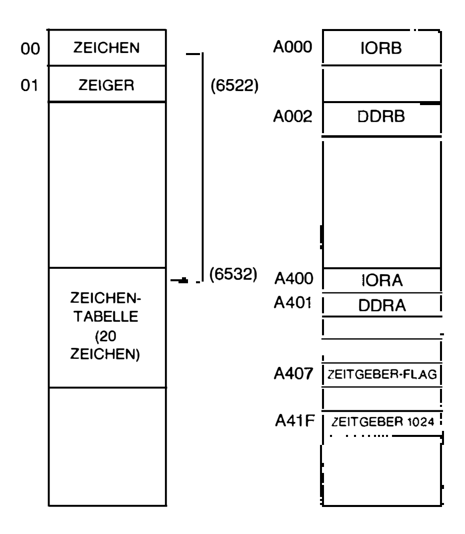


Bild 6.21: Druckerprogramm, Speicherbelegung

```
0200  A9  3F          PRINT   LDA   #$3F     DDRA = „00111111“
0202  8D  01  A4              STA   DDRA
0205  A0  01                  LDY   #$01     Sende Startsignal
0207  8C  00  A0              STY   IORB
020A  88                      DEY
020B  8C  00  A0              STY   IORB     Setze Startsignal zurück
020E  2C  00  A4      TST1    BIT   IORA     Lies Status
0211  70  FB                  BVS   TST1     Zeichenanforderung?
0213  B1  00                  LDA   ($00),Y  Lade Zeichen
0215  8D  00  A4              STA   IORA     Drucke es aus
0218  2C  00  A4      TST2    BIT   IORA     Teste Status
021B  50  FB                  BVC   TST2     Zeichenanforderung gelöscht?
021D  C8                      INY            Nächstes Zeichen
021E  C0  14                  CPY   #$14     20-tes Zeichen?
```

Bild 6.22: Printerprogramm (Programm 6.3)

```
0220  00   EC                   BNE    TST1
0222  A9   20                   LDA    #$20     Lade „space“
0224  8D   00   A4              STA    IORA
0227  A9   30                   LDA    #$30     Zeitkonstante
0229  8D   1F   A4              STA    T1024    Zeitgeber x1024
022C  2C   07   A4   TTIM       BIT    TIMFLG   Zeitgeberstatus?
022F  10   FB                   BPL    TTIM
0231  60                        RTS
0000  50   00                   WORD            Puffer
0050  30   31/3233/34           Puffer BYTE     '0,'1,'2,'3,'4,'5,'6,'7,
      35/3637/3839/40                           '8,'9,'W,'A,'B,'C,'D,
      41/4243/4445/46                           'E,'F,'G,'H,'I
      47/4849
```

Bild 6.22: (Fortsetzung)

Zuerst wird das Datenrichtungsregister A initialisiert:

```
PRINT        LDA  #$3F        „00111111“
             STA  DDRA
```

Dann wird ein Startimpuls erzeugt, indem „00000001“ binär in das IORB geladen wird:

```
             LDA  #$01        „00000001“
             STY  IORB
```

Anschließend wird der Startimpuls wieder zurückgesetzt:

```
             DEY              Y = “00000000“
             STY  IORB
```

Anschließend müssen wir die „Zeichen-Anorderung“-Leitung testen. Solange sie auf „1“ liegt, wiederholen wir die Abfrage. Sobald sie „0“ wird, laden wir das nächste Zeichen:

```
TST1         BIT  IORA        LIES STATUS
             BVS  TST1
```

Wir wollen daran erinnern, daß der BIT-Befehl einen gegebenen Speicherinhalt prüft, ohne ihn zu verändern. Durch diesen Befehl werden die Bits 6 und 7 in das „V“ bzw. in das „N“-Flag kopiert. In unserem Fall soll das Bit 6 geprüft werden (siehe Anschlußbild für Printer, Bild 6.19). Der Befehl BVS prüft das Überlauf-Flag „V“, welches auf denselben Wert wie

Bit 6 des IORB gesetzt worden war. Sein Wert entspricht daher dem Wert auf der „Zeichen-Anfrage"-Leitung. Aus der 20-Zeichen-Tabelle, die ab der Speicheradresse, die in den Speichern „00" und „01" steht, abgespeichert ist, wird nun das nächste Zeichen geladen. Durch einen indirekt adressierten Befehl würde nur das erste Zeichen der Tabelle geladen. Um das Programm allgemeiner zu halten, soll dieser Programmteil in der Lage sein, auf jede Eintragung in der Tabelle zugreifen zu können. Wir werden daher, wie bei jeder Tabellen-Organisation, eine *indizierte Adressierung* anwenden. Als Index-Register nehmen wir das Y-Register. Zu Anfang hat es den Wert „00", der bis zum Wert „19" dezimal inkrementiert wird, bevor die Schleife wieder verlassen wird. Der verwendete Befehl ist indirekt indiziert adressiert:

LDA ($00),Y

Der indirekte indizierte Zugriff wird in Bild 6.23 näher dargestellt. Der Inhalt der Speicher 00/01 wird als Anfangsadresse der Tabelle, auf die zugegriffen werden soll, verwendet. Zu dieser Adresse wird der Inhalt des Y-Registers addiert. Diese endgültige Adresse ist die Adresse des zu ladenden Datenwortes (siehe Bild 6.21). Dieses Datenwort ist der ASCII-Code des Zeichens, das gedruckt werden soll. Er wird zum IORA geschickt:

STA IORA

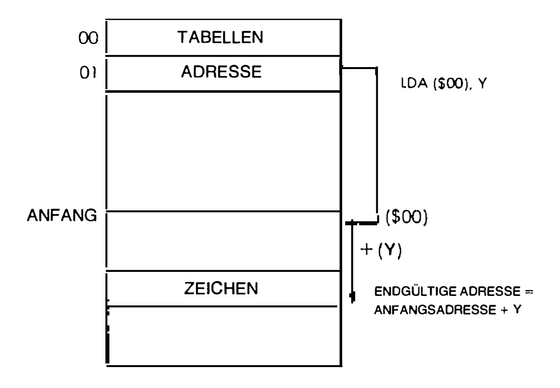


Bild 6.23: Indirekt indizierte Adressierung

Nachdem das Zeichen an den Printer geschickt worden ist, müssen wir warten, bis auf der „Zeichen-Anfrage"-Leitung wieder eine „1" liegt. Ge-

nau wie oben verwenden wir dazu eine Schleife aus zwei Befehlen:

```
TST2        BIT  IORA
            BVC  TST2
```

Nun wird der Zeichenzähler (das Y-Register) inkrementiert:

```
            INY
```

und auf den Grenzwert „20“ dezimal = „14“ hexadezimal geprüft. Solange dieser Grenzwert noch nicht erreicht ist, springen wir wieder an den Schleifenanfang zurück:

```
            CPY  #$14
            BNE  TST1
```

Dann wird der Kode für das abschließende „space“-Zeichen über IORA ausgegeben:

```
            LDA  #$20
            STA  IORA
```

Zum Schluß müssen wir zwischen zwei aufeinanderfolgenden Zeilen eine minimale Verzögerungszeit einhalten. Für den Zeitgeber verwenden wir den Modus „1024-fach“. Diese abschließende Verzögerungszeit von etwa 48 msec erreichen wir, indem wir das entsprechende Zeitgeberregister mit einer Konstanten laden, die die gewünschte Verzögerungszeit in msec angibt (siehe Bild 6.19, Registerverzeichnis):

```
            LDA  #$30      DEZIMAL = 48
            STA  T1024
```

Dann wird das Zeitgeber-Flag solange getestet, bis es „1“ wird, und damit das Ablaufen der programmierten Verzögerung anzeigt:

```
TTIM        BIT  TIMFLG
            BPL  TTIM
```

Bild 6.24 zeigt den Ausdruck der im Beispiel angegebenen 20-Zeichen-Zeile:

```
0123456789@ABCDEFGHI
```

Bild 6.24: Ausdruck der 20-Zeichen-Zeile

Übungsaufgabe 6.10: *Schließen Sie gleichzeitig Printer und Lochstreifenleser an. Der Printer soll den Inhalt des Lochstreifens nach jeder Zeile ausdrucken.*

Zusammenfassung

In diesem Kapitel wurden reale Peripheriegeräte sowohl hardware- als auch softwaremäßig an die Mikrocomputerplatine angeschlossen. Um die Programme zu optimieren, wurden die spezifischen Möglichkeiten der PIO-Register und der Adressierungsarten des 6502 voll ausgenutzt. Der Leser sollte sich nun alle Fähigkeiten angeeignet haben, die für die Erstellung eigener Anwenderprogramme in den häufigsten Fällen benötigt werden.

Kapitel 7

Schlußbetrachtung

Im Rahmen dieses Buches wurde systematisch in die Hardware- und Softwaretechniken eingeführt, die zum Anschließen eines 6502-Mikrocomputers an seine Umwelt benötigt werden. Zuerst wurden die Ein/Ausgabebausteine zusammen mit einigen gängigen 6502-Mikrocomputern beschrieben. Dann wurden in den Kapiteln 4, 5 und 6 Anwenderprogramme mit steigender Komplexität vorgestellt. An diesem Punkt sollte sich der Leser zutrauen können, seinen eigenen 6502-Mikrocomputer an die üblichen Ein/Ausgabebausteine anschließen und die damit verbundenen Hardware- und Softwareprobleme der Schnittstellen lösen zu können. Der Autor ist der Meinung, daß der Leser mit den bis hierher erlangten Fähigkeiten in der Lage sein sollte, praktisch alle Anwendungsprobleme durchschnittlicher Komplexität zu lösen. Natürlich gibt es Fälle, bei denen ganz spezielle Interfaceprobleme existieren. In solchen Fällen sei der Leser auf unser Buch „Mikroprozessor Interface Techniken" verwiesen. Sollte der Leser, der bis zu diesem Punkt gelangt ist, die Übungsaufgaben übersprungen haben, sei ihm dringend empfohlen, zu den Kapiteln 4, 5 und 6 zurückzugehen und alle in diesen Kapiteln vorgeschlagenen Übungsaufgaben zuerst auf dem Papier und dann mit einem richtigen Mikrocomputer zu lösen.

Der nächste Schritt

Falls Sie bis jetzt noch keine Anwenderplatine gebaut haben, ist der nächste logische Schritt, zu Ihrem Elektronikhändler zu gehen, und für ein paar Mark die für die hier vorgeschlagenen Anwendungen benötigten Bauteile zu kaufen. Sie sollten dann versuchen, ohne in diesem Buch nachzuschlagen, selbst einige Programme zu schreiben, und sich zu verge-

wissern, daß Sie auch wirklich alles erlernt haben, was zur Lösung dieser Probleme nötig ist. Mit ein wenig Phantasie können Sie viele andere Anwendungsmöglichkeiten erfinden. Sie können dafür dieselben wenigen Bauteile, oder andere zusätzliche Ein/Ausgabebausteine verwenden.

Für denjenigen Leser, dessen Interesse komplexeren Programmiertechniken gilt, die für die Durchführung schwierigerer Algorithmen benötigt werden, ist in dieser Serie ein dritter Band erschienen, der „Advanced 6502 Programming" heißt. In diesem Buch werden sehr viel komplexere Algorithmen eingeführt und beschrieben. Anhand einer Vielzahl von Spielen, und anderen Programmen, angefangen mit Gedächtnisspielen bis hin zu magischen Quadraten, werden fortgeschrittene Programmierungstechniken vorgestellt.

Es hat sich gezeigt, daß die Zeit, die man zum Erlernen des Programmierens benötigt, von Mensch zu Mensch ganz unterschiedlich ist. Jedoch sollte der nächste logische Schritt nach dem Lesen eines Programmierbuches stets derselbe sein: die Praxis. Es ist die Hoffnung des Autors, daß dieses Buch Ihnen die Fähigkeiten für diesen Schritt in eine erfolgreiche Praxis gewiesen hat.

Anhang A

Ein 6502 Assembler in BASIC

Einleitung

Kurze Programme für den 6502 kann man noch von Hand auf einem Stück Papier entwickeln und dann in den Computer eingeben. Sollen jedoch längere Programme (mit mehr als einigen Dutzend Befehlen), oder eine größere Anzahl kleinerer Programme entwickelt werden, so bietet ein Assembler entscheidende Vorteile. Man kann davon ausgehen, daß die meisten Leser, die ernsthaft daran interessiert sind, ihren 6502 Computer für ihre Anwendungen einzusetzten, mit der Entwicklung solcher Programme beginnen werden. Für diejenigen Leser, die nicht schon anderswo Zugriff auf einen 6502-Assembler haben, bringen wir daher in diesem Anhang A das vollständige Listing eines 6502-Assemblers in BASIC.

Die Programmiersprache BASIC für den Assembler bietet den Vorteil, daß das Programm auf jedem mit BASIC ausgerüsteten Computer läuft. Da insbesondere die Heimcomputer, die fast ausschließlich mit BASIC arbeiten, in letzter Zeit eine enorme Verbreitung gefunden haben, sollte sich für jeden interessierten Anwender Zugriff auf einen solchen Computer finden. Für dieses Programm wurde die von HewlettPackard-Computern verwendete BASIC-Version zugrunde gelegt. Es handelt sich bei diesem BASIC um eine Teilmenge der meisten Mikrocomputer-BASICS, insofern es Erweiterungen neuerer BASIC-Versionen nicht enthält. Will man den Assembler auf einem Computer mit einer anderen BASIC-Version laufen lassen, so ist ein Umschreiben nötig. Da die meisten anderen BASIC-Versionen jedoch wesentlich mehr Möglichkeiten bieten, als das zugrundegelegte BASIC, sollte das Übersetzen des Programms in ein anderes BASIC nicht allzu schwierig sein. Dieser Assembler ist daher im wesentlichen aufwärts kompatibel. Ein Leser, der mit „seinem“ BASIC

gut vertraut ist, wird sogar eine deutliche Straffung des Assemblerprogramms durchführen können.

Mit dem beschriebenen Assembler wurden schon viele Programme für den 6502 assembliert und er hat sich bewährt. So weit uns bekannt ist, ist das Programm fehlerfrei und zuverlässig. Jedoch wird es in diesem Buch nur für Lehrzwecke dargestellt. Für andere Anwendungen kann keinerlei Garantie oder Haftung übernommen werden.

Das Listing des Assemblers ist vollständig. Zur Demonstration der Bedienung wird ein Beispiel vorgeführt.

Alle Programme am Ende von Kapitel 4 wurden mit diesem Assembler erstellt.

Allgemeine Beschreibung

ASM65 ist ein vollständiger mnemonischer Assembler. Er erkennt alle Standard-Mnemonics und erstellt Standard-Listings in hexadezimalem Format wie im Beispiel in Bild A1.

Ferner verfügt der Assembler über alle Standard-Anweisungen außer dem „.“, der aktuelle Speicherzuweisungen und Referenzen kennzeichnet. Es sind folgende Anweisungen möglich: .BYT, .WORD, .DBYT, .TEXT. Die Anwendung dieser Anweisungen kann der Leser in jeder Anleitung für andere Assembler nachlesen.

Bedienung des Assemblers

Der ASM65-Assembler ist in Hewlett-Packard 2000 Serie F BASIC geschrieben. Eine Beschreibung dieser speziellen BASIC-Version wird weiter unten gegeben. Für die Anpassung dieses Assemblers an andere BASIC-Versionen, auf die der Leser Zugriff hat, sollten nur wenige Änderungen nötig sein.

Der ASM65 arbeitet mit sequentiellen Files. Es werden mindestens drei Files benötigt, i. a. jedoch vier. Diese vier Files sind: Quellen-File, Symbol-Tabelle, ein temporäres Hilfs-File, und eventuell ein Ziel-File, das vom Quellen-File verschieden ist.

Das Eingabe-File enthält die Anweisungen in Assembler. Es enthält also Text im ASCII-Format, und muß entsprechend den Regeln der Assembler-Syntax strukturiert sein. Diese Syntax wird im nächsten Abschnitt erläutert. Grundsätzlich können die Eingaben im freien Format geschrieben sein, wenn nur die einzelnen Felder durch ein oder mehrere Leerzeichen getrennt sind. Eine Marke jedoch muß in der ersten Spalte beginnen. Jede Zeile ohne eine Marke darf nicht mit der ersten Spalte beginnen.

```
% CAT QUELL
;PROGRAMM ZUM SPEICHERTRANSFER.
;BIS ZU 255 BYTE EINER TABELLE MIT BEGINN
;BEI LOC1 WERDEN IN EINE TABELLE MIT
;BEGINN BEI LOC2 UEBERTRAGEN. DIE LAENGE
;DES ZU UEBERTRAGENDEN SPEICHERBEREICHS
;IST MOVLEN.
MOVLEN  =$00
LOC1    =$200
LOC2    =$300
;
        LDX MOVLEN  ;LADE TABELLENLAENGE ALS INDEX
LOOP    LDA LOC1,X  ;LADE ZU UEBERTRAGENDES BYTE
        STA LOC2,X  ;SPEICHERE ES IN ZIELTABELLE
        DEX         ;DEKREMENTIERE ZAEHLER X
        BPL LOOP    ;FALLS NICHT ENDE: NAECHSTES BYTE
        RTS         ;FERTIG
% RUN ASM65
QUELLEN FILE ?QUELL
OBJEKT FILE ?OBJ
AUSDRUCK ?JA
ASSEMBLIERUNG BEGINNT...
                ;PROGRAMM ZUM SPEICHERTRANSFER
                ;BIS ZU 255 BYTE EINER TABELLE MIT BEGINN
                ;BEI LOC1 WERDEN IN EINE TABELLE MIT
                ;BEGINN BEI LOC2 UEBERTRAGEN. DIE LAENGE
                ;DES ZU UEBERTRAGENDEN SPEICHERBEREICHS
                ;IST MOVLEN.
                MOVLEN  =$00
                LOC1    =$200
                LOC2    =$300
                ;
0000: A6 00             LDX MOVLEN    ;LADE TABELLENLAENGE ALS INDEX
0002: BD 00 02  LOOP    LDA LOC1,X  ;LADE ZU UEBERTRAGENDES BYTE
0005: 9D 00 03          STA LOC2,X  ;SPEICHERE ES IN ZIELTABELLE
0008: CA                DEX         ;DEKREMENTIERE ZAEHLER
0009: 10 F7             BPL LOOP    ;FALLS NICHT ENDE: NAECHSTES BYTE
000B: 60                RTS         ;FERTIG


SYMBOL TABELLE:
 MOVLEN    0000        LOC1     0200        LOC2     0300
 LOOP      0002
```

Bild A1: Bedienung des ASM65-Assemblers

Der Assembler wird das Kommentarfeld des Ausgabefiles automatisch formatieren. Die anderen Felder mit den Befehlen hingegen werden nicht formatiert. Der Benutzer kann seine Eingaben daher zur besseren Übersicht beliebig tabulieren. Die Lesbarkeit der Programme wird hierdurch gesteigert.

Auch das Ausgabe-File ist ASCII-Text, einschließlich der Darstellung aller Zahlen. Nach dem zweiten Durchlauf des Assemblers kann dieses Ausgabefile wahlweise ausgedruckt werden. Auf die Frage „PRINT-OUT?" im Listing bzw. auf dem Bildschirm antwortet der Benutzer mit „YES" (=„JA") bzw. mit „NO" (=„NEIN").

Der Assembler liefert umfangreiche Fehlermeldungen, beschreibt alle gefundenen Fehler, und listet sie bei der Ausgabe auf.

Die Fehlermeldung des Programms können verschiedene Feldmarkierungen enthalten, wie etwa Operator-Feld-Begrenzer („!"), oder nicht aufgelöste Referenzen („**").

Die Symboltabelle stellt für alle im Programm verwendeten Marken die normale hexadezimale Darstellung auf. Ein Beispiel ist in Bild A2 gezeigt.

```
SYMBOL TABLE:
 MOVLEN     0000            LOC1         0200          LOC2         0300
 LOOP       0002
DONE
```

Bild A2: Die Symboltabelle

Die Syntax

Konstanten

Konstanten können in einem der vier folgenden Formate dargestellt werden:

1. Hexadezimal: der Konstanten muß ein $ vorangestellt sein.
   Beispiel: „LDA $20" lädt den Inhalt des Speichers „20" hexadezimal in den Akkumulator.
2. Binär: der Konstanten muß ein % vorangestellt sein.
   Beispiel: „LDA %11111111" lädt den Akkumulator mit lauter Einsen.
3. Dezimal: Übliche Darstellung.
   Beispiel: „LDA #0" lädt den Akkumulator mit dezimal „0".
4. ASCII: der Konstanten muß ein „'" vorangestellt sein.
   Beispiel: „LDA 'A" lädt den ASCII-Kode für den Buchstaben „A" in den Akkumulator.

Arithmetische Ausdrücke

Arithmetische Ausdrücke können im Operator-Feld, bei einer Marken-Zuweisung, oder in einem Speicherzuweisungsbefehl stehen.

Der Operand in einem arithmetischen Ausdruck kann eine Zahl in einem der vier möglichen Formate sein, oder eine Marke, oder ein „." (das Symbol für die aktuelle Adresse). Kombinationen dieser Möglichkeiten sind erlaubt. Als Operatoren sind "+" und „–" erlaubt. Wird mehr als ein Operator verwendet, so wird der arithmetische Ausdruck von links nach rechts bearbeitet.

Kommentare

Kommentaren muß ein „;" vorangestellt sein. Sie können in jeder Spalte beginnen, einschließlich Spalte 1. Alle Kommentare werden linksbündig in der Mitte des Ausdrucks angeordnet, sofern sie nicht in Spalte 1 beginnen.

Speicherzuweisungen

Speicherzuweisungen werden mit einer der folgenden vier Anweisungen durchgeführt:

.BYT – weist einem Speicherplatz ein Datenbyte zu.
.WORD – weist zwei aufeinanderfolgenden Speicherplätzen zwei Byte in der Reihenfolge niederwertig/höherwertig zu.
.DBYT – weistzwei aufeinanderfolgenden Speicherplätzen zwei Byte in der Reihenfolge höherwertig/niederwertig zu.
.TEXT – wandelt einen Text aus mehreren ASCII-Zeichen in hexadezimale Daten um, die nacheinander in die folgenden Speicherplätze geschrieben werden. Der Text wird durch zweimal dasselbe Zeichen (kein Leerzeichen) beendet.

Es gibt keine end-Anweisung. Stattdessen wird end-of-file benutzt.

Ein Beispiel für eine Speicherzuweisung:

```
.BYT   $2A,WORDCONST
.WORD  2,%10
```

HP 2000 F BASIC

Das BASIC von Hewlett-Packard unterscheidet sich vom BASIC vieler anderer, weit verbreiteter Mini- und Mikrocomputer, aber es läßt sich diesem leicht anpassen. Im folgenden geben wir eine Aufstellung derjenigen Befehle, die anders als bei den meisten anderen BASIC's oder als beim Dartmouth Standard sind.

Files

Files werden in einem FILES-Statement am Anfang eines Programms in der Reihenfolge vereinbart, in der sie im Programm benutzt werden. Das ASSIGN-Statement weist in einem File mit dem ersten Argument einen Namen und mit dem zweiten Argument eine File-Nummer zu. Die Variable im dritten Argument ist bedeutungslos. Ein Sternchen in einem FILES-Statement bedeutet, daß das File erst später durch eine ASSIGN-Anweisung dieser File-Position zugewiesen wird. Die READ-Anweisung liest ein File. Im ersten Argument, dem ein „#“ vorangestellt ist, steht die File-Nummer des Files, das gelesen werden soll. Falls die folgende Satznummer eine Eins ist, und dieser kein Semikolon folgt, so bewirkt diese Anweisung, daß der File-Zeiger auf Null zurückgesetzt wird, z. B.: „READ #2,1“. Alle Argumente hinter dem Semikolon bezeichnen die Variablen, die eingelesen werden sollen.

Die PRINT-Anweisung ähnelt der READ-Anweisung. Ferner gibt es die Sonderform „PRINT #2,END“, mit der eine end-of-file-Marke in das File geschrieben wird.

Die Anweisung IF END # THEN arbeitet ähnlich wie ein Vektor-Interrupt. Wenn beim Lesen eines Files mit READ ein end-of-file vorgefunden wird, führt das Programm einen Sprung zu der Zeilennummer durch, die dem THEN folgt, ohne abzubrechen. Dieser Sprung wird auch dann durchgeführt, wenn der Computer dieses Statement im Moment gerade nicht bearbeitet: d. h. der end-of-file-Vektor braucht im Programm nur einmal gesetzt zu werden, außer man will ihn ändern.

Zeichenketten

Zeichenketten (engl. strings) sind eindimensional und können auch nur eindimensional vereinbart werden. Um einer Zeichenkette die Länge Null zuzuweisen, oder um sie zu löschen, verwendet man die Anweisung L$=„“. Teile einer Zeichenkette können folgendermaßen angesprochen werden: um eine Teil-Kette einer längeren Zeichenkette zu bilden, wird die Anweisung „T$(a,b)“ benutzt. Die Ausdrücke a und b bezeichnen jeweils die erste bzw. letzte Adresse der gewünschten Teil-Kette innerhalb der Haupt-Zeichenkette. Die Adressen innerhalb einer solchen Haupt-Zeichenkette wachsen von links nach rechts, der erste Buchstabe hat die Adresse 1. Beispiel: Wenn A$=„ABCDE“ ist, dann weist der Befehl „B$=A$(2,3)“ der Zeichenkette-Variablen B$ die Zeichenkette „BC“ zu.

Mit dem Ausdruck „T$(a)“ werden alle Zeichen der Zeichenkette T$, beginnend mit dem a-ten Buchstaben bis zum letzten, angesprochen. Beispiel: Wenn A$=„12345“, dann bedeutet A$(3) die Teil-Kette „345“.

Die Zeichenketten-Funktionen CHR$ und ASC, mit denen einen dezimale ASCII-Zahl in die entspreichende Ein-Zeichen-Kette, bzw. umge-

kehrt, umgewandelt wird, stehen nicht zur Verfügung. Das Programm ASM65 liest daher eine Zeichenkette sortierter ASCII-Zeichen von einem System-File mit dem Namen $ASCIIF, der dann zur Umwandlung von Zahlen in Zeichenketten und umgekehrt verwendet wird.

MAX bestimmt die größere von zwei Zahlen. Beispiel: „B=11MAX9" liefert für B den Wert 11.

MIN bestimmt entsprechend die kleinere von zwei Zahlen.

LIN in einer PRINT-Anweisung bewirkt so viele Zeilenvorschübe, wie das folgende Argument angibt.

Die obigen Definitionen sollen als Leitfaden für die Übesetzung des Programmes ASM65 in andere BASIC-Versionen dienen.

Bild A.3: Listing des 6502 Assemblers ASM65 copyright © 1979, Sybex Inc.

```
ASM65

10   REM : ********** 6502 MNEMONIC ASSEMBLER, VERSION 2.0 **********
20   REM
30   REM : GESCHRIEBEN IN HP2000F TSS BASIC
40   REM : KANN FÜR ALLE  65XX PROZESSOREN VON COMMODORE, SYNERTEK
50   REM : UND ROCKWELL VERWENDET WERDEN.
60   REM : ALLE MNEMONICS UND ANWEISUNGEN ENTSPRECHEN DEM INDUSTRIE-
70   REM : STANDARD, AUSSER DEM '.' FÜR DIE LAUFENDE ADRESSE.
80   R=10
90   T9=0
100  A=0
110  DIM L$[72],M$[72],O$[72],C$[72],Z$[72],P$[72],T$[72]
120  DIM A$[72],N$[72]
130  DIM I$[72]
140  L=0
150  FILES *,SYMTAB,TEMP,*,$ASCIIF
160  PRINT "QUELLEN FILE ";
170  PRINT "OBJEKT FILE ";
180  PRINT 'OBJECT FILE ';
190  INPUT O$
200  ASSIGN T$,1,Q8
210  ASSIGN O$,4,Q8
220  READ #1,1
230  PRINT #2,1
240  PRINT #3,1
250  R8=0
260  PRINT "AUSDRUCK ";
270  INPUT I$
280  IF I$ <> "NEIN" THEN 300
290  R8=1
300  PRINT "ASSEMBLIERUNG BEGINNT..."
310  C=0
320  IF  END #1 THEN 2440
330  L$=''
340  I$=''
350  M$=''
360  O$=''
370  C$=''
380  Z$='
```

```
390  L=L+1
400  REM +++++++++ SEPARATE ZEICHEN, SPEICHERE LABELZUWEISUNGEN ++++++++
410  READ #1;I$
420  T5=C
430  IF I$='' THEN 830
440  P=1
450  P$=';'
460  GOSUB 3970
470  IF P1=0 THEN 510
480  IF P1=1 THEN 800
490  C$=I$[P1]
500  I$=I$[1,P1-1]
510  IF I$[1,1]=' ' THEN 590
520  GOSUB 3790
530  L$=P$
540  IF L$ <> '.' THEN 590
550  M$='.'
560  GOSUB 4940
570  L$=''
580  GOTO 860
590  GOSUB 3790
600  M$=P$
610  IF M$[1,3]='.WO' THEN 3110
620  IF M$[1,3]='.TE' THEN 3110
630  IF M$[1,3]='.BY' THEN 3110
640  IF M$[1,3]='.DB' THEN 3110
650  IF M$ <> '' THEN 850
660  C$=C$[1,34]
670  IF LEN(L$) <> 0 THEN 700
680  I$=I$[1,19]
690  GOTO 820
700  GOSUB 3790
710  N$=P$
720  IF LEN(N$) <> 0 THEN 750
730  T1=C
740  GOTO 780
750  GOSUB 4070
760  IF T4=2 THEN 830
770  T1=F1
780  PRINT #2;L$,T1
790  PRINT #2; END
800  I$=I$[1,LEN(I$) MIN 55]
810  Z$[17,17+LEN(I$)]=I$
820  Z$[(LEN(I$)+19 MAX 38) MIN 72]=C$
830  PRINT #3;Z$,T5
840  GOTO 320
850  IF M$[1,1] <> '.' THEN 1050
860  P$='='
870  GOSUB 3970
880  IF P1>0 THEN 910
890  PRINT "ES FEHLT EIN '=' IN ZEILE ";L
900  GOTO 3090
910  P=P1+1
920  GOSUB 3790
930  IF P$[1,1] <> '' THEN 960
940  PRINT "ES FEHLT DAS ARGUMENT IN ZEILE ";L
950  GOTO 3090
960  N$=P$
970  GOSUB 4070
980  IF T4 <> 2 THEN 1010
990  PRINT "UNERLAUBTE VORWÄRTS-REFERENZ IN ZEILE ";L
1000 GOTO 3090
1010 T1=C
1020 C=F1
1030 IF L$ <> '' THEN 780
1040 GOTO 800
```

```
1050  RESTORE 5710
1060  IF M$='' THEN 1140
1070  FOR I=1 TO 56
1080  READ T$
1090  IF T$=M$ THEN 1130
1100  NEXT I
1110  PRINT "UNBEKANNTER OPCODE IN ZEILE ";L
1120  GOTO 3090
1130  O=I
1140  IF L$='' THEN 1170
1150  PRINT #2;L$,C
1160  PRINT #2; END
1170  GOSUB 3750
1180  O$=P$
1190  I$[P-LEN(O$)-1,P-LEN(O$)-1]='!'
1200  REM++++++ FINDE ADRESSIERUNGSART, LADE EFFEKTIVE ADRESSE +++++++
1210  IF O$ <> '' THEN 1240
1220  M=1
1230  GOTO 2200
1240  IF O$ <> 'A' THEN 1270
1250  M=2
1260  GOTO 2200
1270  IF O$[1,1] <> '#' THEN 1320
1280  M=3
1290  P=P+1
1300  N$=O$[2]
1310  GOTO 1870
1320  IF M$[1,1] <> 'B' THEN 1460
1330  IF M$='BIT' THEN 1460
1340  M=12
1350  N$=O$
1360  GOSUB 4070
1370  IF T4 <> 2 THEN 1400
1380  A=-200
1390  GOTO 1970
1400  A=F1-C-2
1410  IF A >= 0 THEN 1430
1420  A=256+A
1430  IF ABS(F1-C) <= 127 THEN 1970
1440  PRINT "UNERLAUBTER SPRUNG IN ZEILE ";L
1450  GOTO 3090
1460  P$='('
1470  P=P-LEN(O$)
1480  GOSUB 3970
1490  P5=P1
1500  P$=','
1510  GOSUB 3970
1520  P6=P1
1530  P7=0
1540  IF  NOT P6 THEN 1610
1550  IF I$[P6+1,P6+1]    'X' THEN 1580
1560  P7=1
1570  GOTO 1610
1580  IF I$[P6+1,P6+1]='Y' THEN 1610
1590  PRINT "UNERLAUBTE ADRESSIERUNGSART IN ZEILE ";L
1600  GOTO 3090
1610  IF P5 <> 0 THEN 1780
1620  GOSUB 3790
1630  N$=P$
1640  IF  NOT P6 OR  NOT P7 THEN 1670
1650  M=5
1660  GOTO 1710
1670  IF  NOT P6 THEN 1700
1680  M=6
1690  GOTO 1710
1700  M=4
```

```
1710  GOSUB 4070
1720  A=F1
1730  IF T4 <> 2 THEN 1750
1740  A=-1000
1750  IF ABS(A) <= 255 THEN 1970
1760  M=M+3
1770  GOTO 1970
1780  GOSUB 3790
1790  N$=P$[2]
1800  IF  NOT P6 OR  NOT P7 THEN 1830
1810  M=10
1820  GOTO 1870
1830  IF  NOT P6 THEN 1860
1840  M=11
1850  GOTO 1870
1860  M=13
1870  GOSUB 4070
1880  A=F1
1890  IF (M <> 10 AND M <> 11) OR A  - 255 THEN 1920
1900  PRINT "WERT ZU GROSS FÜR ZEROPAGE IN ZEILE ";L
1910  GOTO 3090
1920  IF T4 <> 2 THEN 1970
1930  A=-1000
1940  IF M=13 THEN 1970
1950  A=-200
1960  REM+++++++++ SCHREIBE OPCODES & EA AUF FILE +++++++++
1970  IF A >= 0 THEN 2070
1980  Z$[10,11]='**'
1990  C=C+1
2000  IF M <> 12 THEN 2020
2010  Z$[11,11]='R'
2020  W9=A+256
2030  IF W9 >= 0 THEN 2200
2040  Z$[13,14]='**'
2050  C=C+1
2060  GOTO 2200
2070  R=16
2080  I=A
2090  GOSUB 4940
2100  T$=A$
2110  A$='000'
2120  A$[4]=T$
2130  IF (M >= 3 AND M <= 6) OR (M >= 10 AND M  = 12) THEN 2180
2140  Z$[13,14]=A$[LEN(A$)-3,LEN(A$)-2]
2150  Z$[10,11]=A$[LEN(A$)-1]
2160  C=C+2
2170  GOTO 2200
2180  Z$[10,11]=A$[LEN(A$)-1]
2190  C=C+1
2200  R=16
2210  I=T5
2220  GOSUB 4940
2230  T$='000'
2240  T$[4]=A$
2250  Z$[1,4]=T$[LEN(T$)-3]
2260  RESTORE 5140
2270  FOR I=1 TO (O-1)*13+M
2280  READ T$
2290  NEXT I
2300  IF T$ <> '   ' THEN 2370
2310  IF M>6 OR M<4 THEN 2350
2320  M=M+3
2330  C=T5
2340  GOTO 1970
2350  PRINT "UNERLAUBTE ADRESSIERUNGSART IN ZEILE ";L
2360  GOTO 3090
```

```
2370  Z$[7,8]=T$
2380  Z$[5,5]=':'
2390  C=C+1
2400  Z$[17,17+LEN(I$)]=I$
2410  Z$[(19+LEN(I$)) MAX 38]=C$[1,72-(19+LEN(I$) MAX 38)]
2420  PRINT #3;Z$,T5
2430  GOTO 320
2440  REM+++++++++ ZWEITER DURCHLAUF: AUFLÖSEN VON VORWÄRTSREFERENZEN +++++
2450  PRINT #2; END
2460  PRINT #3; END
2470  READ #2,1
2480  L=0
2490  READ #3,1
2500  PRINT #4,1
2510  IF  END #3 THEN 2870
2520  P=1
2530  READ #3;I$,T5
2540  L=L+1
2550  IF I$='' THEN 2850
2560  P$='!'
2570  GOSUB 3970
2580  IF P1=0 OR P1=17 THEN 2610
2590  P=P1
2600  I$[P,P]=' '
2610  IF I$[10,10] <> '*' THEN 2850
2620  GOSUB 3790
2630  N$=P$
2640  IF N$[1,1] <> '(' THEN 2660
2650  N$=N$[2]
2660  GOSUB 4070
2670  IF T4 <> 2 THEN 2700
2680  PRINT "UNLÖSBARE VORWÄRTSREFERENZ / SCHLECHTER LABEL IN ZEILE ";L
2690  GOTO 3090
2700  I=F1
2710  IF I$[11,11] <> 'R' THEN 2750
2720  I=F1-T5-2
2730  IF I >= 0 THEN 2750
2740  I=I+256
2750  R=16
2760  GOSUB 4940
2770  T$=A$
2780  A$='000'
2790  A$[4]=T$
2800  IF I$[13,14] <> '**' THEN 2840
2810  I$[13,14]=A$[LEN(A$)-3,LEN(A$)-1]
2820  I$[10,11]=A$[LEN(A$)-1]
2830  GOTO 2850
2840  I$[10,11]=A$[LEN(A$)-1]
2850  PRINT #4;I$
2860  GOTO 2510
2870  PRINT #4; END
2880  IF R8=1 THEN 3080
2890  IF  END #4 THEN 2940
2900  READ #4,1
2910  READ #4;I$
2920  PRINT I$
2930  GOTO 2910
2940  READ #2,1
2950  PRINT "SYMBOL TABELLE:"
2960  IF  END #2 THEN 3080
2970  FOR I6=1 TO 3
2980  READ #2;O$,T5
2990  R=16
3000  I=T5
3010  GOSUB 4940
3020  T$='0000'
```

```
3030  T$[LEN(T$)+1]=A$
3040  PRINT TAB((I6-1)*25+1);O$;TAB((I6-1)*25+13);T$[LEN(T$)-3];
3050  NEXT I6
3060  PRINT
3070  GOTO 2970
3080  END
3090  PRINT "<"I$">"
3100  END
3110  REM+++++++++ SPEICHERVERWALTUNG +++++++++
3120  Q7=1
3130  IF M$[2,3] <> "TE" THEN 3260
3140  IF Q7 <> 1 THEN 3190
3150  GOSUB 3750
3160  P=P-LEN(P$)
3170  O$=I$[P,P]
3180  P=P+1
3190  IF P <= 72 THEN 3220
3200  PRINT "FALSCHER DELIMITER IN ZEILE ";L
3210  GOTO 3090
3220  P$[1]="'"
3230  P$[2,2]=I$[P,P]
3240  IF P$[2,2]=O$ THEN 320
3250  GOTO 3280
3260  GOSUB 3790
3270  Z$="
3280  P=P+1
3290  IF LEN(P$)=0 THEN 320
3300  N$=P$
3310  GOSUB 4070
3320  IF T4 <> 2 THEN 3350
3330  PRINT "FALSCHES LABEL IN SPEICHERZUWEISUNG VON ZEILE ";L
3340  GOTO 3090
3350  R=16
3360  I=F1
3370  GOSUB 4940
3380  T$=A$
3390  A$="000"
3400  A$[4]=T$
3410  IF M$[2,2] <> "W" THEN 3460
3420  Z$[10,11]=A$[LEN(A$)-3,LEN(A$)-2]
3430  Z$[7,8]=A$[LEN(A$)-1]
3440  C=C+2
3450  GOTO 3560
3460  IF M$[2,2]="D" THEN 3530
3470  IF F1<256 THEN 3500
3480  PRINT "ZU GROSSE ZAHL IN SPEICHERZUWEISUNG VON ZEILE ";L
3490  GOTO 3090
3500  Z$[7,8]=A$[LEN(A$)-1]
3510  C=C+1
3520  GOTO 3560
3530  Z$[7,8]=A$[LEN(A$)-3,LEN(A$)-2]
3540  Z$[10,11]=A$[LEN(A$)-1]
3550  C=C+1
3560  I=T5
3570  R=16
3580  GOSUB 4940
3590  T$="000"
3600  T$[4]=A$
3610  Z$[1,4]=T$[LEN(T$)-3]
3620  Z$[5,5]=":"
3630  IF Q7 <> 1 THEN 3700
3640  IF LEN(L$)=0 THEN 3670
3650  PRINT #2;L$,T5
3660  PRINT #2; END
3670  Z$[17,17+LEN(I$)]=I$
3680  Z$[(19+LEN(I$)) MAX 38]=C$[1,72-(19+LEN(I$)) MAX 38]
```

```
3690  GOTO 3710
3700  Z$=Z$[1,15]
3710  Q7=0
3720  PRINT #3;Z$,T5
3730  T5=C
3740  GOTO 3130
3750  REM++++++++ ROUTINE ZUM ISOLIEREN EINZELNER ZEICHEN ++++++++
3760  REM : SUCHE NACH ZEICHEN BEGINNT BEI P, ZEICHEN WIRD IN P$
3770  REM : GESPEICHERT. P WIRD INKREMENTIERT. FALLS EINSPRUNG HIER,
3775  REM : ENDET SUCHE BEI ' '..
3780  T9=1
3790  REM : FALLS EINSPRUNG HIER, ABBRUCH BEI ' ',',',')','='.
3800  FOR I1=P TO LEN(I$)
3810  IF I$[I1,I1] <> ' ' THEN 3830
3820  NEXT I1
3830  P$=''
3840  FOR I2=I1 TO LEN(I$)
3850  IF I$[I2,I2]=' ' THEN 3920
3860  IF T9=1 THEN 3900
3870  IF I$[I2,I2]=',' THEN 3920
3880  IF I$[I2,I2]=')' THEN 3920
3890  IF I$[I2,I2]='=' THEN 3920
3900  P$[LEN(P$)+1]=I$[I2,I2]
3910  NEXT I2
3920  P=I2
3930  IF LEN(P$)     0 THEN 3950
3940  P=P+1
3950  T9=0
3960  RETURN
3970  REM +++++ FINDE SYMBOL ROUTINE +++++
3980  REM : KOMMT MIT P1=SYMLOC ZURÜCK, WENN GEFUNDEN, UND MIT
3990  REM : P1=0, FALLS NICHT GEFUNDEN.
4000  FOR I=P TO LEN(I$)
4010  IF I$[I,I]=P$[1,1] THEN 4050
4020  NEXT I
4030  P1=0
4040  RETURN
4050  P1=I
4060  RETURN
4070  REM +++++ NUMERISCHER STRING INTERPRETER +++++
4080  REM : VEREINFACHT STRINGS AUS LABELS UND NUMERISCHE AUSDRÜCKE
4090  REM : AUS ZAHLEN IN BELIEBIGEN ZAHLENSYSTEMEN, SOWIE ASCII-KONSTANTEN.
4100  F1=W=0
4110  A$=''
4120  FOR I=1 TO LEN(N$)
4130  IF N$[I,I]='+' THEN 4180
4140  IF N$[I,I]='-' THEN 4180
4150  IF N$[I,I]=')' THEN 4610
4160  A$[LEN(A$)+1]=N$[I,I]
4170  NEXT I
4180  IF A$ <> '.' THEN 4210
4190  F2=C
4200  GOTO 4480
4210  IF A$[1,1]>'Z' THEN 4350
4220  IF A$[1,1]<'A' THEN 4350
4230  READ #2,1
4240  IF  END #2 THEN 4330
4250  READ #2;T$,T1
4260  IF T$ <> A$ THEN 4240
4270  F2=T1
4280  T4=3
4290  IF  END #2 THEN 4320
4300  READ #2;T$,T1
4310  GOTO 4300
4320  GOTO 4480
4330  T4=2
```

```
4340   RETURN
4350   IF A$[1,1] <> '´' THEN 4390
4360   A$=A$[2]
4370   GOSUB 4640
4380   GOTO 4480
4390   B=10
4400   IF A$[1,1] <> '%' THEN 4430
4410   B=2
4420   GOTO 4450
4430   IF A$[1,1] <> '$' THEN 4460
4440   B=16
4450   A$=A$[2]
4460   GOSUB 4750
4470   F2=F
4480   IF W=2 THEN 4510
4490   F1=F1+F2
4500   GOTO 4520
4510   F1=F1-F2
4520   IF I >= LEN(N$) THEN 4610
4530   T$='+-'
4540   FOR W=1 TO LEN(T$)
4550   IF T$[W,W]=N$[I,I] THEN 4590
4560   NEXT W
4570   PRINT "UNERLAUBTER OPERATOR IN ZEILE ";L
4580   GOTO 3090
4590   A$=''
4600   GOTO 4170
4610   T4=0
4620   RETURN
4630   REM +++++ UMWANDLUNG ASCII IN ZAHL +++++
4640   A$=A$[1,1]
4650   F2=0
4660   READ #5,1
4670   READ #5;T$
4680   FOR I=1 TO 72
4690   IF A$[1,1]=T$[I,I] THEN 4740
4700   F2=F2+1
4710   NEXT I
4720   F2=F2-8
4730   GOTO 4670
4740   RETURN
4750   REM +++++ UMWANDLUNG STRING IN ZAHL +++++
4760   REM : B IST DIE BASIS DER ZAHL IN A$, F IST PRODUKT
4770   F=0
4780   I1=0
4790   FOR I2=LEN(A$) TO 1 STEP -1
4800   RESTORE 4910
4810   FOR N=0 TO B-1
4820   READ F$
4830   IF F$=A$[I2,I2] THEN 4870
4840   NEXT N
4850   PRINT "FALSCHE ZAHL IN ZEILE ";L
4860   GOTO 3090
4870   F=F+N*B^I1
4880   I1=I1+1
4890   NEXT I2
4900   RETURN
4910   DATA '0','1','2','3','4','5','6','7','8','9','A','B','C','D'
4920   DATA 'E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S'
4930   DATA 'T','U','V','W','X','Y','Z'
4940   REM +++++ UMWANDLUNG ZAHL IN STRING +++++
4950   REM : I IST INPUT NUMMER, R IST BASIS DER ZAHL IN A$ +++++
4960   A$=''
4970   T=I
4980   FOR N=20 TO 0 STEP -1
4990   IF T/R^N >= 1 THEN 5020
```

```
5000  NEXT N
5010  N=N-1
5020  Q=INT(T/R^N)
5030  IF Q <= R-1 THEN 5050
5040  Q=0
5050  T=T-Q*R^N
5060  RESTORE 4910
5070  FOR S=0 TO Q
5080  READ T$
5090  NEXT S
5100  A$[LEN(A$)+1]=T$
5110  IF N>0 THEN 5010
5120  RETURN
5130  REM +++++ OPCODE TABELLE +++++
5140  DATA '  ','  ','69','65','75','  ','6D','7D','79','61','71','  ','  '
5150  DATA '  ','  ','29','25','35','  ','2D','3D','39','21','31','  ','  '
5160  DATA '  ','0A','  ','06','16','  ','0E','1E','  ','  ','  ','  ','  '
5170  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','90','  '
5180  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','B0','  '
5190  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','F0','  '
5200  DATA '  ','  ','  ','24','  ','  ','2C','  ','  ','  ','  ','  ','  '
5210  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','30','  '
5220  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','D0','  '
5230  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','10','  '
5240  DATA '00','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5250  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','50','  '
5260  DATA '  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','70','  '
5270  DATA '18','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5280  DATA 'D8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5290  DATA '58','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5300  DATA 'B8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5310  DATA '  ','  ','C9','C5','D5','  ','CD','DD','D9','C1','D1','  ','  '
5320  DATA '  ','  ','E0','E4','  ','  ','EC','  ','  ','  ','  ','  ','  '
5330  DATA '  ','  ','C0','C4','  ','  ','CC','  ','  ','  ','  ','  ','  '
5340  DATA '  ','  ','  ','C6','D6','  ','CE','DE','  ','  ','  ','  ','  '
5350  DATA 'CA','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5360  DATA '88','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5370  DATA '  ','  ','49','45','55','  ','4D','5D','59','41','51','  ','  '
5380  DATA '  ','  ','  ','E6','F6','  ','EE','FE','  ','  ','  ','  ','  '
5390  DATA 'E8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5400  DATA 'C8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5410  DATA '  ','  ','  ','  ','  ','  ','4C','  ','  ','  ','  ','  ','6C'
5420  DATA '  ','  ','  ','  ','  ','  ','20','  ','  ','  ','  ','  ','  '
5430  DATA '  ','  ','A9','A5','B5','  ','AD','BD','B9','A1','B1','  ','  '
5440  DATA '  ','  ','A2','A6','  ','B6','AE','  ','BE','  ','  ','  ','  '
5450  DATA '  ','  ','A0','A4','B4','  ','AC','BC','  ','  ','  ','  ','  '
5460  DATA '  ','4A','  ','46','56','  ','4E','5E','  ','  ','  ','  ','  '
5470  DATA 'EA','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5480  DATA '  ','  ','09','05','15','  ','0D','1D','19','01','11','  ','  '
5490  DATA '48','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5500  DATA '08','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5510  DATA '68','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5520  DATA '28','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5530  DATA '  ','2A','  ','26','36','  ','2E','3E','  ','  ','  ','  ','  '
5540  DATA '  ','6A','  ','66','76','  ','6E','7E','  ','  ','  ','  ','  '
5550  DATA '40','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5560  DATA '60','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5570  DATA '  ','  ','E9','E5','F5','  ','ED','FD','F9','E1','F1','  ','  '
5580  DATA '38','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5590  DATA 'F8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5600  DATA '78','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5610  DATA '  ','  ','  ','85','95','  ','8D','9D','99','81','91','  ','  '
5620  DATA '  ','  ','  ','86','  ','96','8E','  ','  ','  ','  ','  ','  '
5630  DATA '  ','  ','  ','84','94','  ','8C','  ','  ','  ','  ','  ','  '
5640  DATA 'AA','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
5650  DATA 'A8','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  '
```

```
5660  DATA 'BA','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','
5670  DATA '8A','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','
5680  DATA '9A','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','
5690  DATA '98','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','  ','
5700  REM +++++ TABELLE DER MNEMONICS +++++
5710  DATA 'ADC','AND','ASL','BCC','BCS','BEQ','BIT','BMI','BNE'
5720  DATA 'BPL','BRK','BVC','BVS','CLC','CLD','CLI','CLV','CMP','CPX','CPY
5730  DATA 'DEC','DEX','DEY','EOR','INC','INX','INY','JMP','JSR','LDA','LDX
5740  DATA 'LDY','LSR','NOP','ORA','PHA','PHP','PLA','PLP','ROL','ROR','RTI
5750  DATA 'RTS','SBC','SEC','SED','SEI','STA','STX','STY','TAX','TAY','TSX
5760  DATA 'TXA','TXS','TYA'
5770  END
```

Anhang B

Kleines Einmaleins: Das Programm

```
                    ;***** EINMALEINS *****
                    ;
                    N       =$00
                    P       =$01
                    NSAVE   =$02
                    PSAVE   =$03
                    T       =$04
                    D       =$9D
                    X       =$200
                    Y       =$201
                    RESUL   =$202
                    ASAVE   =$240
                    XSAVE   =$241
                    YSAVE   =$242
                    PA      =$1700
                    PAD     =$1701
                    TIMER   =$1707
                    ;
                            .=$20
0020: A5 00         START   LDA N
0022: 85 02                 STA NSAVE
0024: A5 01                 LDA P
0026: 85 03                 STA PSAVE
0028: A9 01                 LDA #$01
002A: 8D 01 17              STA PAD
002D: 20 50 02      M1      JSR SOUND
0030: 20 90 00              JSR DL250
0033: C6 00                 DEC N
0035: D0 F6                 BNE M1
0037: A2 14                 LDX #$14            ;2 SEKUNDEN
0039: 20 9C 00              JSR TIME10          ;0,1 SEK UNTERPROGRAMM
003C: 20 50 02      M2      JSR SOUND
003F: 20 90 00              JSR DL250
0042: C6 01                 DEC P
0044: D0 F6                 BNE M2
0046: A9 00         AGAIN   LDA #0
0048: 85 04                 STA T
004A: AD 00 17      POL     LDA PA
```

```
004D: 30 FB             BMI POL
004F: E6 04    PLUS1    INC T           ;TASTE GEDRUECKT?
0051: AD 00 17 M3       LDA PA
0054: 10 FB             BPL M3
0056: A0 1F             LDY #$1F        ;TASTE OFFEN?
0058: A2 01    M4       LDX #1
005A: 20 9E 00          JSR TIME10
005D: AD 00 17          LDA PA
0060: 10 ED             BPL PLUS1
0062: 88                DEY
0063: 10 F3             BPL M4
               ;ANTWORT VOLLSTAENDIG: ERGEBNIS IN T
0065: A6 02             LDX NSAVE
0067: A4 03             LDY PSAVE
0069: 20 10 02          JSR MULTI       ;ERGEBNIS IN A
006C: C5 04             CMP T
006E: F0 0D             BEQ BRAVO
               ;FALSCHE ANTWORT
0070: A0 10             LDY #$10
0072: 20 50 02 M5       JSR SOUND
0075: 20 90 00          JSR DL250
0078: 88                DEY
0079: D0 F7             BNE M5
007B: F0 C9             BEQ AGAIN
               ;RICHTIGE ANTWORT
007D: A0 20    BRAVO    LDY #$20
007F: 20 50 02 M6       JSR SOUND
0082: 88                DEY
0083: D0 FA             BNE M6
0085: 00                BRK
               ;
                        .=$90
0090: 98       DL250    TYA
0091: A2 3D             LDX #$3D
0093: A0 00    DL2      LDY #0
0095: C8       DL1      INY
0096: D0 FD             BNE DL1
0098: E8                INX
0099: D0 F8             BNE DL2
009B: A8                TAY
009C: 60                RTS
               ;
                        .=$9E
009E: 86 9D    TIME10   STX D           ;DAUER IN 1/10 SEC
00A0: A9 62    T0       LDA #$62        ;98 DEZIMAL
00A2: 8D 07 17          STA TIMER       ;ZEIT MAL 1024
00A5: AD 07 17 T1       LDA TIMER
00A8: 10 FB             BPL T1
00AA: C6 9D             DEC D
00AC: D0 F2             BNE T0
00AE: 60                RTS
               ;
                        .=$210
0210: 8E 00 02 MULTI    STX X
0213: 8C 01 02          STY Y
0216: A0 08             LDY #8
0218: A9 00             LDA #0
021A: 4E 00 02 ONE      LSR X
021D: 90 04             BCC TWO
021F: 18                CLC
0220: 6D 01 02          ADC Y
0223: 4A       TWO      LSR A
0224: 6E 02 02          ROR RESUL
0227: 88                DEY
0228: D0 F0             BNE ONE
022A: AD 02 02          LDA RESUL
```

```
022D: 60                    RTS
                  ;
                         .=$250
0250: 8D 40 02   SOUND   STA ASAVE
0253: 8E 41 02           STX XSAVE
0256: 8C 42 02           STY YSAVE
0259: A9 00              LDA #0
025B: A2 80              LDX #$80
025D: A0 00      CL2     LDY #0
025F: C8         CL1     INY
0260: D0 FD              BNE CL1
0262: 49 01              EOR #1
0264: 8D 00 17           STA PA
0267: E8                 INX
0268: D0 F3              BNE CL2
026A: AD 40 02           LDA ASAVE
026D: AE 41 02           LDX XSAVE
0270: AC 42 02           LDY YSAVE
0273: 60                 RTS
```

SYMBOL TABELLE

```
 N          0000      P        0001      NSAVE     0002
 PSAVE      0003      T        0004      D         009D
 X          0200      Y        0201      RESUL     0202
 ASAVE      0240      XSAVE    0241      YSAVE     0242
 PA         1700      PAD      1701      TIMER     1707
 START      0020      M1       002D      M2        003C
 AGAIN      0046      POL      004A      PLUS1     004F
 M3         0051      M4       0058      M5        0072
 BRAVO      007D      M6       007F      DL250     0090
 DL2        0093      DL1      0095      TIME10    009E
 T0         00A0      T1       00A5      MULTI     0210
 ONE        021A      TWO      0223      SOUND     0250
 CL2        025D      CL1      025F
DONE
```

Anhang C

Programm-Listings Aus Kapitel 4, Teil 1

```
ZEILE ADR      CODE       ZEILE

0002  0000               ;DIESES UNTERPROGRAMM NIMMT ASCII-ZEICHEN ZWISCHEN 2CH UND
0003  0000               ;5AH (SOWIE 20H FÜR SPACE) AN UND ERZEUGT DAS ENTSPRECHENDE
0004  0000               ;MORSEZEICHEN MIT EINEM LAUTSPRECHER, DER AN PB7, 6522-U25
0005  0000               ;ANGESCHLOSSEN IST. ES SCHALTET AUSSERDEM PB0 VON 6522-U25
0006  0000               ;EIN UND AUS. MIT EINEM PASSENDEN TREIBER KANN DIESES BIT
0007  0000               ;EINEN SENDER TASTEN. EIN HAUPTPROGRAMM RUFT DIESES UNTER-
0008  0000               ;PROGRAMM MIT DEM ZU SENDENDEN ZEICHEN IM AKKUMULATOR AUF.
0009  0000               ;DAS HAUPTPROGRAMM KÖNNTE Z.B. VON DER TASTATUR EIN ZEICHEN
0010  0000               ;EINLESEN UND MIT DIESEM PROGRAMM DEN MORSECODE AUSSENDEN,
0011  0000               ;ODER MIT EINEM ZUFALLSZAHLENGENERATOR EINE REIHE ZEICHEN ER-
0012  0000               ;ZEUGEN UND DIESE ZUM ÜBEN DES MORSECODES SENDEN. DIE MORSE-
0013  0000               ;ZEICHEN IN DER TABELLE HABEN FOGENDES FORMAT: VON LINKS NACH
0014  0000               ;RECHTS IST DIE ERSTE EINS DAS STARBIT, UND AB DANN BEDEUTET
0015  0000               ;JEDE EINS EINEN STRICH, UND JEDE NULL EINEN PUNKT.
0016  0000               ;
0017  0000               SPEED=$F0
0018  0000               COUNT=$F1
0019  0000               CHAR=$F2
0020  0000                      *=$300
0021  0300: C9 20        MORSE  CMP #$20        ;FALLS SPACE, SPRINGE ZUR SPACEROUTINE
0022  0302: F0 67               BEQ SPACE
0023  0304: C9 2C               CMP #$2C        ;GÜLTIGER ASCII CODE?
0024  0306: 90 4E               BCC EXIT        ;   FALLS KLEINER ALS 2CH: RETURN
0025  0308: C9 5B               CMP #$5B        ;GÜLTIGER ASCII CODE?
0026  030A: B0 4A               BCS EXIT        ;   FALLS GRÖSSER ALS 5AH: RETURN
0027  030C: AA                  TAX             ;SCHIEBE CODE INS X-REGISTER
0028  030D: BD 45 03            LDA TABLE-$2C,X ;HOLE MORSEZEICHEN AUS TABELLE
0029  0310: A0 08               LDY #$8         ;ZAHL DER DURCHZUSCHIEBENDEN BITS
0030  0312: 84 F1               STY COUNT       ;IN ZÄHLREGISTER COUNT
0031  0314: 0A           START8 ASL A
0032  0315: C6 F1               DEC COUNT
0033  0317: 90 FB               BCC START8      ;SCHIEBE A BIS STARTBIT GEFUNDEN
0034  0319: 85 F2               STA CHAR
0035  031B: A5 F2        NEXT   LDA CHAR
0036  031D: 0A                  ASL A           ;AUSGABE MORSECODE (1=STRICH, 0=PUNKT)
0037  031E: 85 F2               STA CHAR
0038  0320: A0 01               LDY #$1         ;PUNKT = 1 VERZÖGERUNGSEINHEIT
0039  0322: 90 02               BCC SEND        ;FALLS CARRY CLEAR: SENDE PUNKT
0040  0324: A0 03               LDY #$3         ;ANDERNFALLS: SENDE STRICH
0041                     ;DIESER TEIL SENDET AM AUSGANG FÜR (Y REGISTER) VERZÖGERUNGS-
0042                     ;EINHEITEN EINE 1, DANN FÜR EINE VERZÖGERUNGSEINHEIT EINE 0.
0043  0326: A9 C0        SEND   LDA #$C0
0044  0328: 8D 0B A0            STA $A00B       ;SETZE ZEITGEBER AUF FREILAUFMODUS
```

Programm 4.1: Morsegenerator (im Text Bild 4.31)

```
0045   0328: A9 00            LOA #Ø0         ;OIESER WERT,
0046   0320: 80 06 A0         STA ØA006
0047   0330: A9 04            LOA #Ø04        ;UNO OIESER WERT, BESTIMMEN OIE TONHÖHE
0048   0332: 80 07 A0         STA ØA007       ;    OES AUSGANGSSIGNALS (CA 1000 HZ)
0049   0335: 80 05 A0         STA ØA005       ;TON EINSCHALTEN
0050   0338: A9 01            LOA #Ø1         ;SCHALTE 8IT P80 AM AUSGANG EIN
0051   033A: 80 00 A0         STA ØA000
0052   0330: 20 57 03         JSR OELAY       ;SPRUNG ZUR VERZÖGERUNGSSCHLEIFE
0053   0340: A9 00            LOA #Ø0
0054   0342: 80 08 A0         STA ØA008       ;SCHALTE TON AUS
0055   0345: 80 00 A0         STA ØA000       ;SCHALTE 8IT P80 AM AUSGANG AUS
0056   0348: A0 01            LOY #Ø01        ;1 VERZÖGERUNGSEINHEIT
0057   034A: 20 57 03         JSR OELAY       ;   FÜR SPACE ZWISCHEN OEN ELEMENTEN
0058   0340: C6 F1            OEC COUNT       ;OEKREMENTIERE ZÄHLER.PRÜFE,O8 8 8IT FERTIG
0059   034F: 00 CA            8NE NEXT        ;   FALLS NEIN: NÄCHSTES ELEMENI
0060   0351: A0 02     FINISH LOY #Ø2         ;VERZÖGERUNG UM WEITERE 2 EINHEITEN
0061   0353: 20 57 03         JSR OELAY       ;   ALS TRENNUNG ZWISCHEN ZEICHEN
0062   0356: 60        EXIT   RTS
0063                   ;VERZÖGERUNG UM (Y REGISTER) X SPEEO X 0.005 SEC
0064   0357: 98        OELAY  TYA
0065   0358: 0A               ASL A
0066   0359: 0A               ASL A
0067   035A: A8               TAY
0068   0358: A5 F0     03     LOA SPEEO
0069   0350: A2 FA     02     LOX #ØFA
0070   035F: CA        01     OEX
0071   0360: 00 F0            8NE 01
0072   0362: 38               SEC
0073   0363: E9 01            S8C #Ø1
0074   0365: 00 F6            8NE 02
0075   0367: 88               OEY
0076   0368: 00 F1            8NE 03
0077   036A: 60               RTS             ;RÜCKSPRUNG AUS VERZÖGERUNGSROUTINE
0078   0368: A0 07     SPACE  LOY #Ø7         ;VERZÖGERUNG UM 7 EINHEITEN
0079   0360: 20 57 03         JSR OELAY       ;   (A8STANO ZWISCHEN WORTEN)
0080   0370: 60               RTS             ;RÜCKSPRUNG AUS MORSEPROGRAMM
0081   0371: 73        TA8LE  .8YTE Ø73,Ø31,Ø6A,Ø32,Ø3F,Ø2F
0081   0372: 31
0081   0373: 6A
0081   0374: 32
0081   0375: 3F
0081   0376: 2F
0082   0377: 27               .8YTE Ø27,Ø23,Ø21,Ø20,Ø30,Ø38
0082   0378: 23
0082   0379: 21
0082   037A: 20
0082   0378: 30
0082   037C: 38
0083   0370: 3C               .8YTE Ø3C,Ø3E,Ø01,Ø01,Ø01,Ø01
0083   037E: 3E
0083   037F: 01
0083   0380: 01
0083   0381: 01
0083   0382: 01
0084   0383: 01               .8YTE Ø01,Ø4C,Ø01,Ø05,Ø18,Ø1A
0084   0384: 4C
0084   0385: 01
0084   0386: 05
0084   0387: 18
0084   0388: 1A
0085   0389: 0C               .8YTE Ø0C,Ø02,Ø12,Ø0E,Ø10,Ø04
0085   038A: 02
0085   0388: 12
0085   038C: 0E
0085   0380: 10
0085   038E: 04
0086   038F: 17               .8YTE Ø17,Ø00,Ø14,Ø07,Ø06,Ø0F
0086   0390: 00
0086   0391: 14
0086   0392: 07
0086   0393: 06
0086   0394: 0F
0087   0395: 16               .8YTE Ø16,Ø10,Ø0A,Ø08,Ø03,Ø09
0087   0396: 10
0087   0397: 0A
0087   0398: 08
0087   0399: 03
0087   039A: 09
0088   0398: 11               .8YTE Ø11,Ø08,Ø19,Ø18,Ø1C
0088   039C: 08
```

Programm 4.1: Morsegenerator (Fortsetzung)

```
0088  0390: 19
0088  039E: 18
0088  039F: 1C


SYMBOL TABELLE:
 SPEED          00F0         COUNT         00F1          CHAR          00F2
 MORSE          0300         STARTB        0314          NEXT          031B
 SEND           0326         FINISH        0351          EXIT          0356
 DELAY          0357         D3            035B          D2            035D
 D1             035F         SPACE         036B          TABLE         0371
```

Programm 4.1: Morsegenerator (Fortsetzung)

```
ZEILE ADR      CODE        ZEILE

0002  0000              ;LADEN SIE ZUERST A7 IN SPEICHER A67E, UND 03 IN SPEICHER A67F.
0003  0000              ;DIES IST EIN UNTERPROGRAMM FÜR EINE ECHTZEITUHR, DIE DIE AKTUELLE
0004  0000              ;UHRZEIT IN DIE SPEICHER SECS(00F6), MIN (00F5), UND HOUR (00F4)
0005  0000              ;SPEICHERT (24 STUNDEN UHR). DURCH DEN INTERRUPT BEIM ABLAUFEN DES
0006  0000              ;ZEITGEBERS WIRD 20 MAL IN DER SEKUNDE EIN SPRUNG ZUM UHRZEITPRO-
0007  0000              ;GRAMM BEWIRKT. ALS ERSTES MÜSSEN DER INTERVALLZEITGEBER UND DAS
0008  0000              ;UHRENPROGRAMM INITIALISIERT WERDEN. DIES MACHT DER TEIL 'INIT'.
0009  0000              ;ZUM STARTEN DES PROGRAMMS MUSS ER ANGESPRUNGEN WERDEN. GEBEN SIE
0010  0000              ;DIE GENAUE UHRZEIT, BEI DER DIE UHR GESTARTET WERDEN SOLL, IN
0011  0000              ;DIE SPEICHER SEC, MIN, HOUR EIN, UND STARTEN SIE DAS PROGRAMM
0012  0000              ;ZU DIESEM ZEITPUNKT MIT 'GO 0390 CR'. MEHR IST NICHT ZU TUN.
0013  0000              COUNT=$00F7               ;ZÄHLER FÜR ZWANZIGSTELSEKUNDEN
0014  0000              SECS=$00F6                ;AKTUELLE UHRZEIT (SEKUNDEN)
0015  0000              MIN=$00F5                 ;                 (MINUTEN)
0016  0000              HOUR=$00F4                ;                 (STUNDEN)
0017  0000              ACR=$A00B                 ;HILFSSTEUERREGISTER FÜR ZEITGEBERMODUS
0018  0000              T1LL=$A006                ;ZEITKONSTANTE, NIEDERWERTIGES BYTE
0019  0000              T1CH=$A005                ;ZEITKONSTANTE, HÖHERWERTIGES BYTE
0020  0000              *=$0390
0021  0390  A9 14       INIT   LDA #$14           ;SETZE ZÄHLER AUF 20
0022  0392  85 F7              STA COUNT
0023  0394  8D 0B A0           STA ACR            ;SETZE BIT 6 UND 7 ACR AUF 0
0024  0397  A9 C0              LDA #$C0           ;SETZE BIT 6 UND 7 IER AUF 1
0025  0399  8D 0E A0           STA $A00E          ;   DAMIT INTERRUPTS ZUGELASSEN SIND
0026  039C  A9 50              LDA #$50           ;LADE ZEITGEBERREGISTER MIT KONSTANTE
0027  039E  8D 06 A0           STA T1LL           ;   C350 (VERZÖGERUNGSKONSTANTE)
0028  03A1  A9 C3              LDA #$C3           ;   ENTSPRECHEND 50 MSEC
0029  03A3  8D 05 A0           STA T1CH           ;HIER WIRD DER ZEITGEBER GESTARTET
0030  03A6  60                 RTS                ;RÜCKSPRUNG IN MONITOR
0031  03A7  08          CLOCK  PHP                ;RETTE STATUSREGISTER
0032  03A8  48                 PHA                ;RETTE AKKUMULATOR
0033  03A9  F8                 SED
0034  03AA  A9 50              LDA #$50           ;LADE ZEITGEBERREGISTER NEU MIT C350
0035  03AC  8D 06 A0           STA T1LL           ;   (VERZÖGERUNGSKONSTANTE FÜR 50 MSEC)
0036  03AF  A9 C3              LDA #$C3
0037  03B1  8D 05 A0           STA T1CH           ;HIER WIRD DER ZEITGEBER GESTARTET
0038  03B4  C6 F7              DEC COUNT          ;DEKREMENTIERE ZÄHLER FÜR 20TEL SEC
0039  03B6  D0 31              BNE EXIT           ;BEI WENIGER ALS 20 ZYKLEN RÜCKSPRUNG
0040  03B8  A9 14              LDA #$14           ;ANDERNFALLS IST 1 SEC VOLL - ZÄHLER
0041  03BA  85 F7              STA COUNT          ;   WIEDER NEU AUF 20 SETZEN
0042  03BC  A9 01              LDA #$01
0043  03BE  18                 CLC
0044  03BF  65 F6              ADC SECS           ;ADDIERE 1 ZU DEN SEKUNDEN
0045  03C1  85 F6              STA SECS
0046  03C3  C9 60              CMP #$60           ;60 SEKUNDEN VOLL?
0047  03C5  D0 22              BNE EXIT           ;FALLS NEIN, RÜCKSPRUNG
0048  03C7  A9 00              LDA #$00           ;ANDERNFALLS SETZE SECS AUF 0 ZURÜCK
0049  03C9  85 F6              STA SECS
0050  03CB  A9 01              LDA #$01
0051  03CD  18                 CLC
0052  03CE  65 F5              ADC MIN            ;ADDIERE 1 ZU DEN MINUTEN
0053  03D0  85 F5              STA MIN
0054  03D2  C9 60              CMP #$60           ;60 MINUTEN VOLL?
0055  03D4  D0 13              BNE EXIT           ;FALLS NEIN, RÜCKSPRUNG
0056  03D6  A9 00              LDA #$00           ;ANDERNFALLS SETZE MIN AUF 0 ZURÜCK
0057  03D8  85 F5              STA MIN
0058  03DA  A9 01              LDA #$01
0059  03DC  18                 CLC
0060  03DD  65 F4              ADC HOUR           ;ADDIERE 1 ZU DEN STUNDEN
0061  03DF  85 F4              STA HOUR
0062  03E1  C9 24              CMP #$24           ;24 STUNDEN VOLL?
0063  03E3  D0 04              BNE EXIT           ;FALLS NEIN, RÜCKSPRUNG
0064  03E5  A9 00              LDA #$00           ;ANDERNFALLS SETZE HOUR AUF 0 ZURÜCK
0065  03E7  85 F4              STA HOUR
0066  03E9  68          EXIT   PLA                ;HOLE AKKUMULATOR ZURÜCK
0067  03EA  28                 PLP                ;HOLE STATUSREGISTER ZURÜCK
0068  03EB  40                 RTI                ;RÜCKSPRUNG VON INTERRUPT


SYMBOL TABELLE

ACR      A00B    CLOCK    03A7    COUNT    00F7    EXIT     03E9
HOUR     00F4    INIT     0390    MIN      00F5    SECS     00F6
T1CH     A005    T1LL     A006
```

Programm 4.2: 24-Stunden-Uhr (im Text Bild 4.37)

```
ZEILE ADR      CODE       ZEILE

0002  0000                ;DIES IST EIN EINFACHES PROGRAMM ZUR HEIM-
0003  0000                ;STEUERUNG, DAS IN EINER ENDLOSSCHLEIFE LÄUFT.
0004  0000                ;BEI JEDEM DURCHLAUF ZEIGT ES DIE AKTUELLE
0005  0000                ;ZEIT AN UND SPRINGT IN EINIGE UNTERPROGRAMME,
0006  0000                ;DIE ANGESCHLOSSENE GERÄTE STEUERN, Z.B.:
0007  0000                ;1) EIN UNTERPROGRAMM KANN DIE UHRZEIT TESTEN
0008  0000                ;   UND ZU EINER BESTIMMTEN ZEIT EINE LAMPE
0009  0000                ;   EIN - ODER AUSSCHALTEN.
0010  0000                ;2) EIN ANDERES UNTERPROGRAMM KÖNNTE DEN STATUS
0011  0000                ;   EINER ALARMANLAGE PRÜFEN UND BEIM AUFTRETEN
0012  0000                ;   EINES ALARMS GEEIGNET REAGIEREN.
0013  0000                DDRB=$AC02
0014  0000                IORB=$AC00
0015  0000                HOUR=$00F4
0016  0000                MIN=$00F5
0017  0000                OUTBYT=$82FA
0018  0000                SCAND=$8906
0019  0000                       *=$0200
0020  0200  D8            CONTRL CLD
0021  0201  A9 0F                LDA #$0F         ;SETZE DATENRICHTUNGS-
0022  0203  8D 02 AC             STA DDRB         ;REGISTER AUF AUSGABE
0023  0206  A9 00                LDA #$00
0024  0208  8D 00 AC             STA IORB         ;SCHALTE RELAIS AUS
0025  020B  A5 F4         LOOP   LDA HOUR         ;HIER HAUPTSCHLEIFE
0026  020D  20 FA 82             JSR OUTBYT       ;DISPLAY ZEIGT STUNDE
0027  0210  A5 F5                LDA MIN
0028  0212  20 FA 82             JSR OUTBYT       ;DISPLAY ZEIGT MINUTE
0029  0215  20 06 89             JSR SCAND        ;DISPLAY AKTIVIEREN
0030  0218  EA                   .BYTE $EA,$EA,$EA
0030  0219  EA
0030  021A  EA
0031  021B  EA                   .BYTE $EA,$EA,$EA
0031  021C  EA
0031  021D  EA
0032  021E  EA                   .BYTE $EA,$EA,$EA
0032  021F  EA
0032  0220  EA
0033  0221  EA                   .BYTE $EA,$EA,$EA
0033  0222  EA
0033  0223  EA
0034  0224  EA                   .BYTE $EA,$EA,$EA
0034  0225  EA                                    ;  HIER KANN DER
0034  0226  EA                                    ;  BENUTZER JUMP-
0035  0227  EA                   .BYTE $EA,$EA,$EA;  BEFEHLE ZU DEN
0035  0228  EA                                    ;  UNTERPROGRAMMEN
0035  0229  EA                                    ;  EINFÜGEN, DIE
0036  022A  EA                   .BYTE $EA,$EA,$EA;  ANGESPRUNGEN WERDEN
0036  022B  EA                                    ;  SOLLEN.
0036  022C  EA
0037  022D  EA                   .BYTE $EA,$EA,$EA
0037  022E  EA
0037  022F  EA
0038  0230  EA                   .BYTE $EA,$EA,$EA
0038  0231  EA
0038  0232  EA
0039  0233  EA                   .BYTE $EA,$EA,$EA
0039  0234  EA
0039  0235  EA
0040  0236  4C 0B 02             JMP LOOP


SYMBOL TABELLE

CONTRL   0200      DDRB    AC02      HOUR    00F4
IORB     AC00      LOOP    020B      MIN     00F5
OUTBYT   82FA      SCAND   8906
```

Programm 4.3: Heimsteuerung (im Text Bild 4.38)

```
ZEILE ADR      CODE       ZEILE

0002  0000                ;DIESES PROGRAMM KANN GESPEICHERTE TELEFONNUMMERN
0003  0000                ;SELBSTSTÄTIG WÄHLEN. ÜBER EINEN LAUTSPRECHER AM
0004  0000                ;AUSGANG ERZEUGT ES EIN ZWEITONSIGNAL. FÜR SCHALTBILD
0005  0000                ;SIEHE Z.B. FIG. 4.45. WENN DER LAUTSPRECHER DIREKT
0006  0000                ;AN DER SPRECHMUSCHEL LIEGT, AKTIVIERT DAS ZWEITON-
0007  0000                ;SIGNAL DIE WÄHLELEKTRONIK. DIE GEWÜNSCHTEN TELEFON-
0008  0000                ;NUMMERN WERDEN IRGENDWO IM SPEICHER ABGELEGT, BYTE
0009  0000                ;FÜR BYTE. DAS LETZTE BYTE MUSS EIN OF SEIN. FÜR DIE
0010  0000                ;TELEFONNUMMER 555-1212 WÜRDE BEISPIELSWEISE DIE
0011  0000                ;BYTEFOLGE 05 05 05 01 02 01 02 0F (ALLES HEX) IM
0012  0000                ;SPEICHER ABGELEGT. SPEICHERN SIE DANN DIE ANFANGS-
0013  0000                ;ADRESSE DER TELEFONNUMMER IN DER REIHENFOLGE LOW
0014  0000                ;HIGH IN DIE SPEICHER 00C0 und 00C1. SPRINGEN SIE
0015  0000                ;DANN DIESES UNTERPROGRAMM ENTWEDER VOM MONITOR
0016  0000                ;ODER VON IRGENDEINEM HAUPTPROGRAMM AN.
0017  0000                NUMPTR=$00C0          ;ZEIGER AUF BEGINN DER TELEFONNUMMER
0018  0000                ONDEL=$40             ;VERZÖGERUNGSKONSTANTE, WENN TÖNE
                                                ;EINGESCHALTET SIND
0019  0000                DELCON=$FF            ;VERZÖGERUNGSKONSTANTE
0020  0000                ACR1=$AD0B            ;MODUS ZEITGEBER 1
0021  0000                ACR2=$AC0B            ;MODUS ZEITGEBER 2
0022  0000                T1CH=$AD05            ;ZÄHLREGISTER (HIGH) VON
                                                ;ZEITGEBER 1
0023  0000                T1LH=$AD07            ;ZWISCHENSPEICHER (HIGH) VON
                                                ;ZEITGEBER 1
0024  0000                T1LL=$AD04            ;ZWISCHENSPEICHER (LOW) VON
                                                ;ZEITGEBER 1
0025  0000                T2CH=$AC05            ;DASSELBE FÜR ZEITGEBER 2
0026  0000                T2LH=$AC07
0027  0000                T2LL=$AC04
0028  0000                OFFDEL=$20            ;VERZÖGERUNGSKONSTANTE, WENN TÖNE
                                                ;AUSGESCHALTET SIND
0029  0000                      *=$0300
0030  0300  A0 00    PHONE  LDY #$00           ;ZÄHLER FÜR Y-TE ZAHL DER TEL NUMMER
0031  0302  B1 C0    DIGIT  LDA(NUMPTR),Y      ;LADE Y-TE ZAHL
0032  0304  C8              INY
0033  0305  C9 0F           CMP #$0F           ;ENDE DER TEL NUMMER?
0034  0307  D0 01           BNE NOEND
0035  0309  60              RTS                ;RÜCKSPRUNG IN MONITOR BZW. IN DAS
                                                ;AUFRUFENDE PROGRAMM
0036  030A  0A EA EA NOEND  ASL A              ;MULTIPLIZIERE ZU WÄHLENDE ZAHL MIT 4
0037  030D  0A EA EA        ASL A              ;ERGIBT INDEX FÜR FREQUENZTABELLE
                                                ;  (JEDER EINTRAG IST 4 BYTE LANG)
0038  0310  AA              TAX                ;X=INDEX FÜR TABELLENZUGRIFF
0039  0311  A9 C0           LDA #$C0
0040  0313  8D 0B AD        STA ACR1           ;BEIDE ZEITGEBER IM FREILAUFMODUS
0041  0316  8D 0B AC        STA ACR2
0042  0319  BD 50 03        LDA TABLE,X        ;LADE ERSTEN TON, LOW
0043  031C  8D 04 AD        STA T1LL           ;SPEICHERE IN ZEITGEBER 1
0044  031F  E8              INX
0045  0320  BD 50 03        LDA TABLE,X        ;LADE ERSTEN TON, HIGH
0046  0323  8D 07 AD        STA T1LH           ;SPEICHERE IN ZEITGEBER 1
0047  0326  8D 05 AD        STA T1CH           ;STARTE ZEITGEBER 1
0048  0329  E8              INX
0049  032A  BD 50 03        LDA TABLE,X        ;ZWEITER TON, LOW BYTE
0050  032D  8D 04 AC        STA T2LL           ;IN ZEITGEBER 2
0051  0330  E8              INX
0052  0331  BD 50 03        LDA TABLE,X        ;ZWEITER TON, HIGH BYTE
0053  0334  8D 07 AC        STA T2LH           ;IN ZEITGEBER 2
0054  0337  8D 05 AC        STA T2CH           ;STARTE ZEITGEBER 2
0055  033A  A2 40           LDX #ONDEL         ;LADE VERZÖGERUNGSKONSTANTE FÜR
                                                ;EINSCHALTPERIODE
0056  033C  20 55 03 ON     JSR DELAY          ;EINSCHALTVERZÖGERUNG
0057  033F  CA              DEX
0058  0340  D0 FA           BNE ON
0059  0342  A9 00           LDA #$00
0060  0344  8D 0B AD        STA ACR1           ;SCHALTE BEIDE ZEITGEBER AUS
0061  0347  8D 0B AC        STA ACR2
0062  034A  A2 20           LDX #OFFDEL        ;LADE VERZÖGERUNGSKONSTANTE FÜR
                                                ;   AUSSCHALTPERIODE
0063  034C  20 55 03 OFF    JSR DELAY          ;AUSSCHALTVERZÖGERUNG
0064  034F  CA              DEX
0065  0350  D0 FA           BNE OFF
0066  0352  4C 02 03        JMP DIGIT          ;SPRINGE AN ANFANG ZURÜCK UND
                                                ;   BEARBEITE NÄCHSTE ZAHL
0067  0355                ;
0068  0355                ;EINFACHE VERZÖGERUNGSROUTINE FÜR EIN- UND AUSSCHALTPERIODE
0069  0355                ;
```

Programm 4.4: Telefonwähler (im Text Bild 4.41)

```
0070  0355  A9 FF     DELAY  LOA #DELCON        ;LADE VERZÖGERUNGSKONSTANTE
0071  0357  38        WAIT   SEC                ;
0072  0358  E9 01            SBC #$01
0073  035A  00 FB            BNE WAIT           ;VERZÖGERUNGSSCHLEIFE
0074  035C  60                  RTS             ;RÜCKSPRUNG AUS ROUTINE DELAY
0075  0350                   ;
0076  0350                   ;DIES IST OIE TABELLE DER KONSTANTEN FÜR OIE TON-
0077  0350                   ;FREQUENZEN, OIE ZU DEN 10 ZIFFERN GEHÖREN. OIE KONSTANTEN
0078  0350                   ;SIND JE 2 BYTE LANG, DAS LOW BYTE STEHT ZUERST.
0079  0350                   ;
0080  0350  13        TABLE  .BYTE $13,$02,$76,$01   ;2 TÖNE FÜR '0'
0080  035E  02
0080  035F  76
0080  0360  01
0081  0361  C0               .BYTE $C0,$02,$9E,$01   ;'1'
0081  0362  02
0081  0363  9E
0081  0364  01
0082  0365  C0               .BYTE $CD,$02,$76,$01   ;'2'
0082  0366  02
0082  0367  76
0082  0368  01
0083  0369  C0               .BYTE $CD,$02,$53,$01   ;'3'
0083  036A  02
0083  0368  53
0083  036C  01
0084  0360  89               .BYTE $89,$02,$9E,$01   ;'4'
0084  036E  02
0084  036F  9E
0084  0370  01
0085  0371  89               .BYTE $89,$02,$76,$01   ;'5'
0085  0372  02
0085  0373  76
0085  0374  01
0086  0375  89               .BYTE $89,$02,$53,$01   ;'6'
0086  0376  02
0086  0377  53
0086  0378  01
0087  0379  48               .BYTE $48,$02,$9E,$01   ;'7'
0087  037A  02
0087  037B  9E
0087  037C  01
0088  0370  48               .BYTE $48,$02,$76,$01   ;'8'
0088  037E  02
0088  037F  76
0088  0380  01
0089  0381  48               .BYTE $48,$02,$53,$01   ;'9'
0089  0382  02
0089  0383  53
0089  0384  01
0090  0385                   .END


SYMBOL TABELLE

ACR1     A008     ACR2    AC0B     DELAY    0355     DELCON   00FF
DIGIT    0302     NDEND   030A     NUMPTR   00C0     OFF      034C
OFFDEL   0020     ON      033C     ONDEL    0040     PHONE    0300
T1CH     A005     T1LH    A007     T1LL     A004     T2CH     AC05
T2LH     AC07     T2LL    AC04     TABLE    0350     WAIT     0357
```

Programm 4.4: Telefonwähler (Fortsetzung)

Anhang D

Hexadezimale Umwandlungstabelle

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEZ	HEX	DEZ	HEX	DEZ	HEX	DEZ	HEX	DEZ	HEX	DEZ
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

Anhang E

ASCII Umwandlungstabelle

HEX		0	1	2	3	4	5	6	7
	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	–	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB		7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	•	·	J	Z	j	z
B	1011	VT	ESC	+	,	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	- -
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	∧	n	~
F	1111	SI	US	/	?	O	←	o	DEL

DIE ASCII SYMBOLE

NUL Null
SOH Beginn des Kopfes (Start of Heading)
STX Beginn des Textes (Start of Text)
ETX Ende des Textes (End of Text)
EOT Ende der Übertragung (End of Transmission)
ENQ Anfrage (Enquing)
ACK Bestätigung (Acknowledge)
BEL Klingel (Bell)
BS Zurücksetzen (Backspace)
HT Horizontaler Tabulator (Horizontal Tabulation)
LF Zeilenvorschub (Line Feed)
VT Vertikaler Tabulator (Vertical Tabulation)
FF Format Vorschub (Form Feed)
CR Wagenrücklauf/Zeilenwechsel (Carriage Return)
SO Rückschaltung (Shift Out)
SI Dauerumschaltung (Shift In)
DLE Datenverbindungs-Umschaltung (Data Link Escape)
DC Gerätesteuerung (Device Control)
NAK Negativ-Bestätigung (Negative Acknowledge)
SYN Synchronisations-Leerlauf (Synchronous Idle)
ETB Ende des Übertragungsblockes (End of Transmission Block)
CAN Annullieren (Cancel)
EM Datenträgerende (End of Medium)
SUB Ersetzen (Substitute)
ESC Umschaltung (Escape)
FS Dateitrennzeichen (File Separator)
GS Gruppentrennzeichen (Group Separator)
RS Satztrennzeichen (Record Separator)
US Einheiten-Trennzeichen (Unit Separator)
SP Leerzeichen (Space/Blank)
DEL Löschzeichen (Delete)

Anhang F

6502 Befehlssatz – Alphabetisch

ADC	Addieren mit Übertrag
AND	Logische UND
ASL	Arithmetisches Linksschieben
BCC	Verzweigen, wenn Übertrag gelöscht
BCS	Verzweigen, wenn Übertrag gesetzt
BEQ	Verzweigen, wenn Result = 0
BIT	Teste Bit
BMI	Verzweigen, wenn Minus
BNE	Verzweigen, wenn ungleich 0
BPL	Verzweigen, wenn Plus
BRK	Abbruch
BVC	Verzweigen, wenn Überlauf gelöscht
BVS	Verzweigen, wenn Überlauf gesetzt
CLC	Übertrag löschen
CLD	Dezimal-Flagge löschen
CLI	Unterbrechungsabschaltung löschen
CLV	Überlauf löschen
CMP	Vergleichen mit Akkumulator
CPX	Vergleichen mit X
CPY	Vergleichen mit Y
DEC	Dekrementieren Speicher
DEX	Dekrementieren X
DEY	Dekrementieren Y
EOR	Exklusives ODER
INC	Inkrementieren Speicher
INX	Inkrementieren X
INY	Inkrementieren Y
JMP	Verzweigen
JSR	Verzweigen in Unterprogramm
LDA	Laden Akkumulator
LDX	Laden X
LDY	Laden Y
LSR	Logisches Rechtsschieben
NOP	Leerbefehl (keine Operation)
ORA	Logisches ODER
PHA	Push A
PHP	Push P Status
PLA	Pop A
PLP	Pop P Status
ROL	Linksrotieren
ROR	Rechtsrotieren
RTI	Rückkehr von Unterbrechung
RTS	Rückkehr aus Unterprogramm
SBC	Subtrahieren mit Übertrag
SEC	Übertrag setzen
SED	Dezimal setzen
SEI	Unterbrechungsabschaltung setzen
STA	Akkumulator speichern
STX	X speichern
STY	Y speichern
TAX	A nach X übertragen
TAY	A nach Y übertragen
TSX	SP (Stapelzeiger) nach X übertragen
TXA	X nach A übertragen
TXS	X nach SP (Stapelzeiger) übertragen
TYA	Y nach A übertragen

Anhang G

6502 Befehlssatz – Binär

ADC	011bbb01	JSR	00100000
AND	001bbb01	LDA	101bbb01
ASL	000bbb10	LDX	101bbb10
BCC	10010000	LDY	101bbb00
BCS	10110000	LSR	010bbb10
BEQ	11110000	NOP	11101010
BIT	0010b100	ORA	000bbb01
BMI	00110000	PHA	01001000
BNE	11010000	PHP	00001000
BPL	00010000	PLA	01101000
BRK	00000000	PLP	00101000
BVC	01010000	ROL	001bbb10
BVS	01110000	ROR	011bbb10
CLC	00011000	RTI	01000000
CLD	11011000	RTS	01100000
CLI	01011000	SBC	111bbb01
CLV	10111000	SEC	00111000
CMP	110bbb01	SED	11111000
CPX	1110bb00	SEI	01111000
CPY	1100bb00	STA	100bbb01
DEC	110bb110	STX	100bb110
DEX	11001010	STY	100bb100
DEY	10001000	TAX	10101010
EOR	010bbb01	TAY	10101000
INC	111bb110	TSX	10111010
INX	11101000	TXA	10001010
INY	11001000	TXS	10011010
JMP	01b01100	TYA	10011000

Anhang H

6502 Befehlssatz: Hexadezimal mit Zeitangaben

		Inplizit			Akkumulator			Absolut			Seite 0			Unmittelbar			Abs X		
Mnemonisch		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ADC	(1)							6D	4	3	65	3	2	69	2	2	7D	4	3
AND	(1)							2D	4	3	25	3	2	29	2	2	3D	4	3
ASL					0A	2	1	0E	6	3	06	5	2				1E	7	3
BCC	(2)																		
BCS	(2)																		
BEQ	(2)																		
BIT								2C	4	3	24	3	2						
BMI	(2)																		
BNE	(2)																		
BPL	(2)																		
BRK		00	7	1															
BVC	(2)																		
BVS	(2)																		
CLC		1B	2	1															
CLD		DB	2	1															
CLI		58	2	1															
CLV		BB	2	1															
CMP								CD	4	3	C5	3	2	C9	2	2	DD	4	3
CPX								EC	4	3	E4	3	2	E0	2	2			
CPY								CC	4	3	C4	3	2	C0	2	2			
DEC								CE	6	3	C6	5	2				DE	7	3
DEX		CA	2	1															
DEY		BB	2	1															
EOR	(1)							4D	4	3	45	3	2	49	2	2	5D	4	3
INC								EE	6	3	E6	5	2				FE	7	3
INX		E8	2	1															
INY		CB	2	1															
JMP								4C	3	3									
JSR								20	6	3									
LDA	(1)							AD	4	3	A5	3	2	A9	2	2	BD	4	3
LDX	(1)							AE	4	3	A6	3	2	A2	2	2			
LDY	(1)							AC	4	3	A4	3	2	A0	2	2	BC	4	3
LSR					4A	2	1	4E	6	3	46	5	2				5E	7	3
NOP		EA	2	1															
ORA								0D	4	3	05	3	2	09	2	2	1D	4	3
PHA		48	3	1															
PHP		08	3	1															
PLA		68	4	1															
PLP		28	4	1															
ROL					2A	2	1	2E	6	3	26	5	2				3E	7	3
ROR					6A	2	1	6E	6	3	66	5	2				[illegible]	7	3
RTI		40	6	1															
RTS		60	6	1															
SBC	(1)							ED	4	3	E5	3	2	E9	2	2	FD	4	3
SEC		38	2	1															
SED		F8	2	1															
SEI		78	2	1															
STA								8D	4	3	85	3	2				9D	5	3
STX								8E	4	3	86	3	2						
STY								8C	4	3	84	3	2						
TAX		AA	2	1															
TAY		A8	2	1															
TSX		BA	2	1															
TXA		8A	2	1															
TXS		9A	2	1															
TYA		98	2	1															

(1) 1 zu n dazuzählen wenn Seitengrenze überschritten wird

Abs Y			(IND. X)			(IND)Y			Z Seite X			Relativ			Indirekt			Z Seite Y			Prozessor Status Kodes							
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N	V		B	D	I	Z	C
79	4	3	61	6	2	71	5	2	75	4	2										●	●					●	●
39	4	3	21	6	2	31	5	2	35	4	2										●						●	
									16	6	2										●						●	●
												90	2	2														
												B0	2	2														
												F0	2	2														
																					M_7	M_6					●	
												30	2	2														
												D0	2	2														
												10	2	2														
																								1		1		
												50	2	2														
												70	2	2														
																												0
																									0			
																										0		
																						0						
D9	4	3	C1	6	2	D1	5	2	D5	4	2										●						●	●
																					●						●	●
																					●						●	●
									D6	6	2										●						●	
																					●						●	
																					●						●	
59	4	3	41	6	2	51	5	2	55	4	2										●						●	
									F6	6	2										●						●	
																					●						●	
																					●						●	
																6C	5	3										
B9	4	3	A1	6	2	B1	5	2	B5	4	2										●						●	
BE	4	3																B6	+	2	●						●	
									B4	4	2										●						●	
									56	6	2										0						●	●
19	4	3	01	6	2	11	5	2	15	4	2										●						●	
																					●						●	
																					●	●	●	●	●	●	●	●
									36	6	2										●						●	●
									76	6	2										●						●	●
																					●	●	●	●	●	●	●	●
F9	4	3	E1	6	2	F1	5	2	F5	4	2										●	●					●	●
																												1
																									1			
																										1		
99	5	3	81	6	2	91	6	2	95	4	2																	
																		96	4	2								
									94	4	2																	
																					●						●	
																					●						●	
																					●						●	
																					●						●	
																					●						●	

(2) 2 zu n dazuzählen wenn in einer Seite gesprungen wird
3 zu n dazuzählen wenn in eine andere Seite gesprungen wird

Stichwortverzeichnis

Die SYBEX Bibliothek

MEIN ERSTER COMPUTER (2. überarbeitete Ausgabe)
von Rodnay Zaks – eine Einführung für alle, die den Kauf oder die Nutzung eines Mikrocomputers erwägen. 305 Seiten, 150 Abbildungen, Format DIN A5, Ref.Nr.: **3020** (1982)

CP/M HANDBUCH MIT MP/M
von Rodnay Zaks – ein umfassendes Lehr- und Nachschlagewerk für CP/M, das Standard Betriebssystem für Mikrocomputer. 310 Seiten, 100 Abbildungen, Format DIN A5, Ref.Nr. **3002** (1981)

MIKROPROZESSOR INTERFACE TECHNIKEN (3. überarbeitete Ausgabe)
von Rodnay Zaks/Austin Lesea – Hardware und Software Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 435 Seiten, 400 Abbildungen, Format DIN A5, Ref.Nr.: **3012** (1982)

PROGRAMMIERUNG DES 6502 (2. überarbeitete Ausgabe)
von Rodnay Zaks – Programmierung in Maschinensprache mit dem Mikroprozessor 6502, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 350 Seiten, 160 Abbildungen, Format DIN A5, Ref.Nr.: **3011** (1982)

PROGRAMMIERUNG DES Z80
von Rodnay Zaks – ein kompletter Lehrgang in der Programmierung des Z80 Mikroprozessors und eine gründliche Einführung in die Maschinensprache. 630 Seiten, 200 Abbildungen, Format DIN A5, Ref.Nr.: **3006** (1982)

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 540 Seiten, 130 Abbildungen, Ref.Nr.: **3004** (1981)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 510 Seiten, 270 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3005** (1982)

PASCAL PROGRAMME FÜR WISSENSCHAFTLER UND INGENIEURE
von Alan Miller – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal Benutzer mit technischen Anwendungen. 320 Seiten, 120 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3007** (1982)

POCKET MIKROCOMPUTER LEXIKON
– die schnelle Informations-Börse! 1300 Definitionen, Kurzformeln, technische Daten, Lieferanten-Adressen, ein englisch-deutsches und französisch-deutsches Wörterbuch. 176 Seiten, Format DIN A6, Ref.Nr. **3008** (1982)

BASIC COMPUTER SPIELE/Band 1
herausgegeben von David H. Ahl – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 59 Abbildungen, Ref.Nr. **3009**

BASIC COMPUTER SPIELE/Band 2
herausgegeben von David H. Ahl – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Ref.Nr.: **3010**

BASIC PROGRAMME FÜR WISSENSCHAFTLER UND INGENIEURE
von Alan Miller – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 320 Seiten, 120 Abbildungen, Ref.Nr.: **3015** (Erscheint Herbst 1983)

EINFÜHRUNG IN DIE TEXTVERARBEITUNG
von H. Glatzer – aus was eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und zu was sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 208 Seiten, 67 Abbildungen, Ref.Nr. **3018** (1983)

EINFÜHRUNG IN WORDSTAR
von A. Naiman – eine klar gegliederte Einführung, die aufzeigt, wie WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 208 Seiten, 30 Abbildungen, Ref.Nr.: **3019** (1983)

6502 ANWENDUNGEN
von Rodnay Zaks – das Eingabe-/Ausgabe-Buch für Ihren 6502-Mikroprozessor. Stellt die meistgenutzten Programme und die dafür notwendigen Hardware-Komponenten vor. 288 Seiten, 205 Abbildungen, Ref.Nr.: **3014** (1983)

BASIC ÜBUNGEN FÜR DEN APPLE
von J.-P. Lamoitier – das Buch für APPLE-Nutzer, die einen schnellen Zugang zur Programmierung in BASIC suchen. Abgestufte Übungen mit zunehmendem Schwierigkeitsgrad. 240 Seiten, 185 Abbildungen, Ref.Nr.: **3016** (1983)

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 560 Seiten, 325 Abbildungen, Ref.Nr.: **3017** (Erscheint Herbst 1983)

VORSICHT! Computer brauchen Pflege
von Rodnay Zaks – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemein gültige Regeln für die pflegliche Behandlung Ihres Systems. 224 Seiten, 96 Abbildungen, Ref.Nr.: **3013** (Erscheint April 1983)

MEIN SINCLAIR ZX81
von D. Hergert – eine gut lesbare Einführung in diesen Einplatincomputer und dessen Programmierung in BASIC. 180 Seiten, 20 Abbildungen, Ref: **3021** (Erscheint Februar 1983)

The SYBEX Library*

** Mikrocomputer-Bücher in englischer Sprache von SYBEX; lieferbar ab Lager Düsseldorf.*

YOUR FIRST COMPUTER

by Rodnay Zaks 264 pp., 150 illustr., Ref. C200A

The most popular introduction to small computers and their peripherals: what they do and how to buy one.

DON'T (or How to Care for Your Computer)

by Rodnay Zaks 222 pp., 100 illust., Ref. C400

The correct way to handle and care for all elements of a computer system, including what to do when something doesn't work.

INTERNATIONAL MICROCOMPUTER DICTIONARY

140 pp., Ref. X2

All the definitions and acronyms of microcomputer jargon defined in a handy pocket-size edition. Includes translations of the most popular terms into ten languages.

FROM CHIPS TO SYSTEMS: AN INTRODUCTION TO MICROPROCESSORS

by Rodnay Zaks 558 pp., 400 illustr. Ref. C201A

A simple and comprehensive introduction to microprocessors from both a hardware and software standpoint: what they are, how they operate, how to assemble them into a complete system.

INTRODUCTION TO WORD PROCESSING

by Hal Glatzer 216 pp., 140 illustr., Ref. W101

Explains in plain language what a word processor can do, how it improves productivity, how to use a word processor and how to buy one wisely.

INTRODUCTION TO WORDSTAR™

by Arthur Naiman 208 pp., 30 illustr., Ref. W105

Makes it easy to learn how to use WordStar, a powerful word processing program for personal computers.

VISICALC® APPLICATIONS

by Stanley R. Trost 200 pp.,-Ref. V104

Presents accounting and management planning applications—from financial statements to master budgets; from pricing models to investment strategies.

EXECUTIVE PLANNING WITH BASIC

by X. T. Bui 192 pp., 19 illust., Ref. B380

An important collection of business management decision models in BASIC, including Inventory Management (EOQ), Critical Path Analysis and PERT, Financial Ratio Analysis, Portfolio Management, and much more.

BASIC FOR BUSINESS

by Douglas Hergert 250 pp., 15 illustr., Ref. B390

A logically organized, no-nonsense introduction to BASIC programming for business applications. Includes many fully-explained accounting programs, and shows you how to write them.

PROGRAMMING THE 6809

by Rodnay Zaks and William Labiak 520 pp., 150 illustr., Ref. C209

This book explains how to program the 6809 in assembly language. No prior programming knowledge required.

PROGRAMMING THE 6502

by Rodnay Zaks 388 pp., 160 illustr., Ref. C202

Assembly language programming for the 6502, from basic concepts to advanced data structures.

6502 APPLICATIONS BOOK

by Rodnay Zaks 286 pp., 200 illustr., Ref. D302

Real-life application techniques: the input/output book for the 6502.

ADVANCED 6502 PROGRAMMING

by Rodnay Zaks 292 pp., 140 illustr., Ref. G402A

Third in the 6502 series. Teaches more advanced programming techniques, using games as a framework for learning.

PROGRAMMING THE Z80

by Rodnay Zaks 626 pp., 200 illustr., Ref. C280

A complete course in programming the Z80 microprocessor and a thorough introduction to assembly language.

PROGRAMMING THE Z8000

by Richard Mateosian 300 pp., 124 illustr., Ref. C281

How to program the Z8000 16-bit microprocessor. Includes a description of the architecture and function of the Z8000 and its family of support chips.

THE CP/M® HANDBOOK (with MP/M™)

by Rodnay Zaks 324 pp., 100 illustr., Ref. C300

An indispensable reference and guide to CP/M—the most widely-used operating system for small computers.

MASTERING CP/M®

by Alan R. Miller 320 pp., Ref. C302

For advanced CP/M users or systems programmers who want maximum use of the CP/M operating system . . . takes up where our *CP/M Handbook* leaves off.

INTRODUCTION TO THE UCSD p-SYSTEM™

by Charles W. Grant and Jon Butah 250 pp., 10 illustr., Ref. P370

A simple, clear introduction to the UCSD Pascal Operating System; for beginners through experienced programmers.

A MICROPROGRAMMED APL IMPLEMENTATION

by Rodnay Zaks 350 pp., Ref. Z10

An expert-level text presenting the complete conceptual analysis and design of an APL interpreter, and actual listing of the microcode.

THE APPLE CONNECTION

by James W. Coffron 228 pp., 120 illustr., Ref. C405

Teaches elementary interfacing and BASIC programming of the Apple for connection to external devices and household appliances.

MICROPROCESSOR INTERFACING TECHNIQUES

by Rodnay Zaks and Austin Lesea 458 pp., 400 illust., Ref. C207

Complete hardware and software interconnect techniques, including D to A conversion, peripherals, standard buses and troubleshooting.

FORDERN SIE EIN GESAMTVERZEICHNIS UNSERER VERLAGSPRODUKTION AN:



6502

Anwendungen ist der Folgeband zu dem Sybex-Standardwerk „Programmierung des 6502".

Im Rahmen dieses Buches werden systematisch die Hardware- und Softwaretechniken vorgestellt, die zum Anschließen eines 6502-Mikrocomputers an seine Umwelt benötigt werden. Sie lernen praktische Anwendungstechniken für Ihren 6502 Mikroprozessor und wie Anwenderprogramme für den 6520, 6522, 6530 und 6532 geschrieben und implementiert werden. Zusätzliche Schnittstellen für einfache Anschlußgeräte werden präsentiert, so daß Sie eine Anwenderplatine zum Üben bauen können. Das Buch beinhaltet eine Reihe von abgestuften Übungen, um Ihre erworbenen Fähigkeiten auf die Probe zu stellen. Die besten Resultate werden Sie erzielen, wenn Sie die Übungen an einem 6502-System durcharbeiten.

Der Autor

hat Kurse in Programmierung über Mikroprozessoranwendungen vor mehreren Tausend Lernwilligen in der Welt gegeben. Er hat seinen Doktor in Computertechnologie von der kalifornischen Universität Berkeley. Er hat bereits einen programmierten APL-Anwendungskurs entwickelt und arbeitete in Silicon Valley in Anwendung und Entwicklung von industriellen Mikroprozessorsystemen, als diese das erste Mal auf den Markt kamen. Er hat einige der Bestseller im Bereich Mikrocomputerbücher auf den Markt gebracht, die jetzt bereits in 10 Sprachen verfügbar sind. Dieses Buch, wie auch die anderen in dieser Serie, sind durch seine Erfahrung mit technischem und erzieherischem Hintergrund geschaffen worden.

ISBN 3-88745-014-0